

Charles Rugg PS04

**Analytical Investigation of the Dynamics
of Tethered Constellations in Earth Orbit (Phase II)**

Contract NAS8-36606

Quarterly Report #13

For the period 1 April 1988 through 30 June 1988

Principal Investigators

Dr. Enrico C. Lorenzini
Dr. Gordon E. Gullahorn
Dr. Robert D. Estes

July 1988

Prepared for
National Aeronautics and Space Administration
Marshall Space Flight Center, Alabama 35812

Smithsonian Institution
Astrophysical Observatory
Cambridge, Massachusetts 02138

The Smithsonian Astrophysical Observatory
is a member of the
Harvard-Smithsonian Center for Astrophysics

(NASA-CR-179371) ANALYTICAL INVESTIGATION
OF THE DYNAMICS OF TETHERED CONSTELLATIONS
IN EARTH ORBIT Quarterly Report No. 13, 1
Apr. - 30 Jun. 1988 (Smithsonian
Astrophysical Observatory) 124 p

N88-28950

Unclas

CSCL 22B G3/18 0156618

**Analytical Investigation of the Dynamics
of Tethered Constellations in Earth Orbit (Phase II)**

Contract NAS8-36606

Quarterly Report #13

For the period 1 April 1988 through 30 June 1988

Principal Investigator
Dr. Enrico C. Lorenzini
Dr. Gordon E. Gullahorn
Dr. Robert D. Estes

Co-Investigators
Dr. Mario D. Grossi
Dr. Mario Cosmo
Mr. David A. Arnold

July 1988

Prepared for
National Aeronautics and Space Administration
Marshall Space Flight Center, Alabama 35812

Smithsonian Institution
Astrophysical Observatory
Cambridge, Massachusetts 02138

The Smithsonian Astrophysical Observatory is a member of the Harvard-Smithsonian Center for Astrophysics
--

CONTENTS

	Page
Summary	1
Figure Captions	3
SECTION 1.0 INTRODUCTION	4
2.0 TECHNICAL ACTIVITY DURING REPORTING PERIOD AND PROGRAM STATUS, E.C. LORENZINI, PI	5
2.1 Wave Propagation In The Upper Tether Of TECS	5
2.1.1 Introductory Remarks	5
2.1.2 Transverse Waves	7
2.1.3 Longitudinal Waves	12
2.1.4 Numerical Results	14
2.1.5 References To Section 2.1	24
2.2 Longitudinal Dampers	25
2.2.1 Introductory Remarks	25
2.2.2 Optimization Of Longitudinal Damper Parameters	26
2.2.3 References To Section 2.2	41
2.3 Concluding Remarks	41
3.0 PROBLEMS ENCOUNTERED DURING REPORT PERIOD, E.C. LORENZINI, PI	43
4.0 ACTIVITY PLANNED FOR NEXT REPORTING PERIOD, E.C. LORENZINI, PI	43
5.0 TECHNICAL ACTIVITY DURING REPORTING PERIOD AND PROGRAM STATUS, G. GULLAHORN PI.	44

CONTENTS (Cont.)

		Page
SECTION 5.1	Tether Applications Simulation Working Group Support	44
5.2	Tether Aerodynamic Effect Of RCS Thruster Plume	45
5.2.1	Computer Acquisition	45
5.2.2	Transport Of SLACK Code To Microcomputer	46
5.2.3	Modification Of SLACK Code	50
6.0	PROBLEMS ENCOUNTERED DURING REPORTING PERIOD, G. GULLAHORN PI.	50
7.0	ACTIVITY PLANNED FOR NEXT REPORTING PERIOD, G. GULLAHORN PI.	51
7.1	Tether Applications Simulation Working Group Support	51
7.2	Tether Aerodynamic Effect Of RCS Thruster Plume	52
8.0	TECHNICAL ACTIVITY DURING REPORTING PERIOD AND PROGRAM STATUS, R.D. ESTES, PI	53
9.0	PROBLEMS ENCOUNTERED DURING REPORTING PERIOD, R.D. ESTES, PI	119
10.0	ACTIVITY PLANNED FOR NEXT REPORTING PERIOD, R.D. ESTES, PI	119

Summary

This Quarterly Report deals with the three sets of topics summarized hereunder. For each set of topics the name of the PI responsible for that particular set is indicated.

(1) E.C. Lorenzini, PI.

Investigation of the propagation of longitudinal and transverse waves along the upper tether. Specifically the upper tether is modelled as three massive platforms connected by two perfectly elastic continua (tether segments). The tether attachment point to the station is assumed to vibrate both longitudinally and transversally at a given frequency. Longitudinal and transverse waves propagate along the tethers affecting the acceleration levels at the elevator and at the upper-platform. The displacement and acceleration frequency-response-functions at the elevator and at the upper-platform are computed for both longitudinal and transverse waves.

An analysis to optimize the damping time of the longitudinal dampers is also carried out in order to select the "optimal" damper parameters. The analytical evaluation of the performance of tuned longitudinal dampers vs. detuned longitudinal dampers is also part of this analysis.

(2) G.E. Gullahorn, PI.

Due to other demands on the time of one PI, reduced effort was spent on the tasks (A) for support of the Tether Applications Simulation Working Group and (B) for study of the use of the Shuttle primary Reaction Control System (RCS) thrusters for blowing away a recoiling broken tether. In support of the latter, a microcomputer system was acquired and set up. SLACK code has been transferred to this system from the VAX, and translated from VAX extended Fortran to the Fortran-77 standard used on the microcomputer.

(3) R.D. Estes, PI.

Most of the effort in the tether plasma physics study was devoted to software development in this period. A particle simulation code has been integrated into our Macintosh II computer system and will be utilized for studying the physics of hollow cathodes.

Figure Captions

- Figure 1. Schematic of the upper-tether.
- Figure 2(a)-2(f). Displacement and acceleration frequency-response-functions (FRF's) at the elevator for longitudinal and transverse waves propagating in the upper-tether.
- Figure 3(a)-3(h). Same as Figures 2 except that the FRF's are those of the upper-platform instead of the elevator. The longitudinal and transverse displacement attenuation-functions for waves crossing the elevator and reaching the upper-platform are shown in Figures 3(g) and 3(h).
- Figure 4. Schematic of a longitudinal damper and associated tether segment.
- Figures 5(a)-5(c). Longitudinal dynamic response vs. damper parameters and tether characteristics of tether segment 1 (between the lower-platform and the station). These figures are also representative of the dynamic response of tether segment 3 (between the elevator and the upper-platform).
- Figures 6(a)-6(c). Same as in Figures 5 except that the dynamic response is that of tether segment 2 (between the station and the elevator).
- Figure 7. First "card" of hypercard front-end to simulation program. User defines the basic run parameters.
- Figure 8. Example of a card to define the input parameters for a given species of simulation particle.

1.0 INTRODUCTION

This is Quarterly Report #13 submitted by the Smithsonian Astrophysical Observatory (SAO) under NASA/MSFC contract NAS8-36606, "Analytical Investigation of the Dynamics of Tethered Constellations in Earth Orbit (Phase II)." The PI's for this report are: Dr. Enrico C. Lorenzini for the analysis described in Sections 2.0-4.0, Dr. Gordon E. Gullahorn for Sections 5.0—7.0, and Dr. Robert D. Estes for Sections 8.0—10.0. This report covers the period from 1 April 1988 through 30 June 1988.

2.0 TECHNICAL ACTIVITY DURING REPORTING PERIOD AND PROGRAM STATUS, E.C. LORENZINI, PI

2.1 Wave Propagation In The Upper Tether Of TECS

2.1.1 Introductory Remarks

An important topic of investigation for the Tether Elevator/Crawler System (TECS) is the propagation of disturbances along the tethers. Specifically, the propagation along the upper tether is of particular relevance to the microgravity experiments on board the elevator (*EL*). The Space Station (*SS*), in fact, is a source of non-negligible disturbances at a wide range of frequencies. In general, the low frequencies (around 10^{-3} Hz) disturbances are associated with aerodynamic and orbital perturbations, the medium frequencies (10^{-2} Hz—10 Hz) disturbances with the structural vibrations of the station, and the higher frequencies (>10 Hz) with rotating machinery and human activity on board the station.

Since the tether is an elastic continuum the above mentioned disturbances travels from the *SS* to the upper-tether-end and back affecting the microgravity level on board the elevator. For a perfectly elastic tether with non-dissipative terminations these waves propagate indefinitely back and forth along the tether. In an actual case the tether has a small amount of material damping and is, moreover, embedded in a dissipative medium. The same applies to the terminations (platforms).

The amount of tether material damping has been estimated in a preliminary way. The tether is a complex non-isotropic continuum and the damping varies with the tether length according to a function which depends on the damping model adopted. Furthermore, material damping is significantly affected by temperature because the material properties of kevlar are sensitive to temperature variations. Current estimate of tether material damping ranges from 1% to 5% of the critical damping for a tether length of 20 km [1,2]. The tether, therefore, has to be viewed as a low-damping elastic continuum.

The small tether material damping affects primarily the propagation of longitudinal waves, while has an almost negligible effect on the transverse waves [1]. Transverse waves, on the other hand, are affected by the interaction with the surrounding atmosphere. The extent to which this interaction provides non-negligible damping of transverse waves along the tether has not been evaluated so far but it is quite reasonable that it is small given the low value of the tether transverse velocity.

The first issue to be addressed, therefore, is the response of the upper tether, considered as a perfectly elastic continuum, when its attachment point to the station is oscillating in either the longitudinal or the transverse direction with a given frequency.

If we consider the realistic case of small perturbations we can treat the longitudinal and the transverse waves independently as shown in the next

subsections.

2.1.2 Transverse Waves

The upper tether subsystem is schematically shown in Figure 1. The tether segment 1 of length ℓ_1 connects the attachment point of the station to the elevator. The tether segment 2 of length ℓ_2 connects the elevator to the upper platform. The longitudinal coordinates along the undeformed tether segments 1 and 2 are called z_1 and z_2 respectively; therefore $0 \leq z_1 \leq \ell_1$ and $0 \leq z_2 \leq \ell_2$. The attachment point at the station is perturbed in either the transverse or the longitudinal direction as indicated in Figure 1. In particular the transverse perturbation generates transverse waves propagating in the two tether segments. We call ψ_1 and ψ_2 the lateral deformations of tether segments 1 and 2 produced by the transverse waves.

After assuming that the platforms are point masses the transverse wave equations in tether segments 1 and 2, and the boundary conditions are given by

$$\frac{\partial^2 \psi_1}{\partial t^2} = c_{\psi_1}^2 \frac{\partial^2 \psi_1}{\partial z_1^2} \quad 0 \leq z_1 \leq \ell_1 \quad (1.1)$$

$$\frac{\partial^2 \psi_2}{\partial t^2} = c_{\psi_2}^2 \frac{\partial^2 \psi_2}{\partial z_2^2} \quad 0 \leq z_2 \leq \ell_2 \quad (1.2)$$

$$\psi_1(0, t) = p_T e^{i\omega t} \quad (1.3)$$

UPPER-TETHER

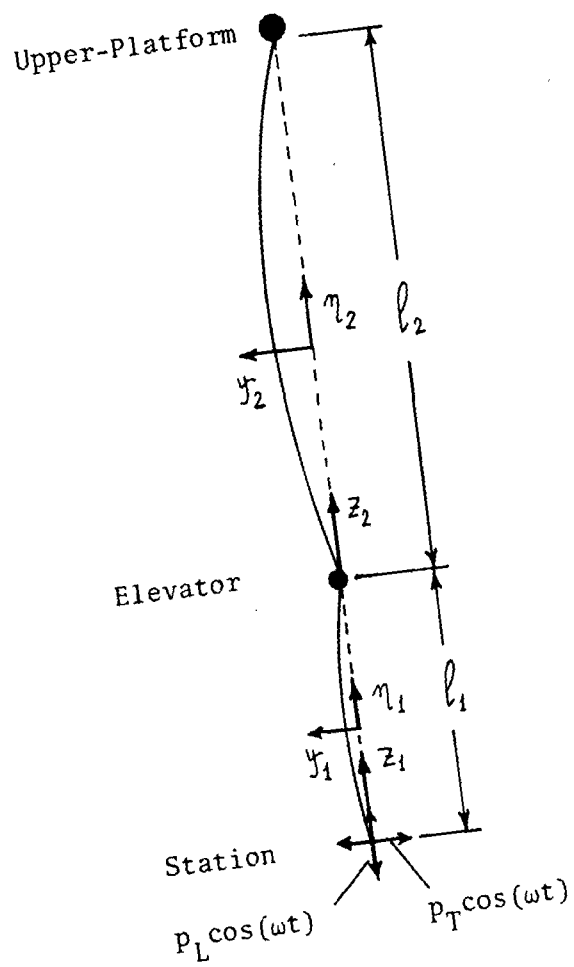


Figure 1

$$-\left[\frac{\partial\psi_1}{\partial z_1}(\ell_1, t)\right]T_1 - \left[\frac{\partial\psi_2}{\partial z_2}(0, t)\right]T_2 = \left[\frac{\partial^2\psi_2}{\partial t^2}(0, t)\right]m_1 \quad (1.4)$$

$$\psi_1(\ell_1, t) = \psi_2(0, t) \quad (1.5)$$

$$-\left[\frac{\partial\psi_2}{\partial z_2}(\ell_2, t)\right]T_2 = \left[\frac{\partial^2\psi_2}{\partial t^2}(\ell_2, t)\right]m_2 \quad (1.6)$$

where we have expressed the external perturbation in equation (1.3) as a complex-variable function by utilizing Euler's formula. The boundary condition of equations (1.4) states that the inertial force at the elevator (m_1) is balanced by the resultant of the transverse forces generated by tether segments 1 and 2. Equation (1.6) is similar to equation (1.4) except that is referred to the upper platform (m_2) which is connected to tether segment 2 only. Equation (1.5) is an interface condition between the transverse displacement of tether segment 1 and 2 at the elevator. The solutions of equations (1.1) and (1.2) have the form

$$\psi_1 = R_1(z_1)e^{i\omega t} \quad (2.1)$$

$$\psi_2 = R_2(z_2)e^{i\omega t} \quad (2.2)$$

After substituting equations (2) into equations (1.1) and (1.2) we obtain standard differential equations in $R_1(z_1)$ and $R_2(z_2)$ whose solutions are

$$R_j(z_j) = \text{Re}[D_j e^{ik_j z_j}] \quad j = 1, 2 \quad (3)$$

where D_j 's are complex numbers which can be expressed as follows

$$D_j = d_j + i\tilde{d}_j \quad j = 1, 2 \quad (4)$$

and

$$k_j = \omega / c_{\psi j} \quad j = 1, 2 \quad (5)$$

We define

$$\phi_j = k_j \ell_j \quad j = 1, 2 \quad (6)$$

and after substituting equations (3) into equations (2) and subsequently into the boundary conditions (1.3)–(1.6) we obtain

$$\begin{aligned} d_1 &= p_T \\ \left(\tilde{d}_1 \cos \phi_1 + d_1 \sin \phi_1 \right) T_1 k_1 + \left(\tilde{d}_2 \cos \phi_2 + d_2 \sin \phi_2 \right) T_2 k_2 &= -\omega^2 m_1 d_2 \\ d_1 \cos \phi_1 - \tilde{d}_1 \sin \phi_1 &= d_2 \\ \left(\tilde{d}_2 \cos \phi_2 + d_2 \sin \phi_2 \right) T_2 k_2 &= -\omega^2 m_2 \left(d_2 \cos \phi_2 - \tilde{d}_2 \sin \phi_2 \right) \end{aligned} \quad (7)$$

By solving equations (7) for d_1 , \tilde{d}_1 , d_2 , and \tilde{d}_2 we can explicit the shape functions $R_j(z_j)$ given by equations (3). In particular we are interested in the values of the function $R_1(z_1)$ and $R(z_2)$ at the coordinates $z_1 = \ell_1$ and $z_2 = \ell_2$ respectively. $R_1(\ell_1)$ and $R_2(\ell_2)$, in fact, provide the transverse displacements at the elevator and at the upper platform respectively for a transverse displacement perturbation of

the tether attachment point to the station.

After several algebraic manipulations we obtain the displacements at the elevator and at the upper platform as follows:

$$R_1(\ell_1) = p_T \chi_1 \quad (7.1)$$

$$R_2(\ell_2) = p_T \chi_2 = p_T \chi_1 \theta_T \quad (7.2)$$

where

$$|\chi_1| = \left| \cos \phi_1 - \omega a_1 \sin \phi_1 + \tau \frac{\omega a_2 \sin \phi_1}{\cos \phi_2 - \omega a_2 \sin \phi_2} \right|^{-1} \quad (8.1)$$

$$|\theta_T| = |\cos \phi_2 - \omega a_2 \sin \phi_2|^{-1} \quad (8.2)$$

$$|\chi_2| = |\chi_1| |\theta_T| \quad (8.3)$$

and

$$a_j = m_j c_{\psi_j} / T_j \quad j = 1, 2 \quad (9.1)$$

$$\phi_j = \omega \ell_j / c_{\psi_j} \quad j = 1, 2 \quad (9.2)$$

$$\tau = T_2 c_{\psi_1} / (T_1 c_{\psi_2}) \quad (9.3)$$

$|\chi_1|$ and $|\chi_2|$ are the transverse-displacement frequency-response-functions (*FRF*) at the elevator and at the upper platform. From equations (7) we infer that $|\theta_T|$ provides the attenuation of a transverse wave passing through the elevator and tether segment 2. It is comforting to notice that for $T_2 = 0$ (no wave propagation in tether 2) equation (8.1) reduces to the form shown in reference [2] which has been derived for a single tether segment. Since the wave equations are linear we

can easily obtain the acceleration *FRF*'s $|\chi_1^a|$ and $|\chi_2^a|$ as follows [1]

$$|\chi_j^a| = \phi_j^2 |\chi_j| \quad j = 1, 2 \quad (10)$$

2.1.3 Longitudinal Waves

By inspecting equations (1) we see that the transverse wave equations and their boundary conditions can be transformed into longitudinal wave equations and associated boundary conditions by making the following substitutions: (a) the transverse displacement ψ_j is substituted by the longitudinal displacement η_j ; (b) the transverse wave velocity c_{ψ_j} by the longitudinal wave velocity c_{η_j} ; (c) the tensions T_j by $(EA)_j$; and (d) the transverse displacement amplitude p_T by the longitudinal displacement amplitude p_L . After the transformation the longitudinal wave equations are obtained as follows:

$$\frac{\partial^2 \eta_j}{\partial t^2} = c_{\eta_j} \frac{\partial^2 \eta_j}{\partial z_j^2} \quad 0 \leq z_j \leq \ell_j, \quad j = 1, 2 \quad (11.1)$$

$$\eta_1(0, t) = p_L e^{i\omega t} \quad (11.2)$$

$$-\left[\frac{\partial \eta_1}{\partial z_1}(\ell_1, t)\right](EA)_1 - \left[\frac{\partial \eta_2}{\partial z_2}(0, t)\right](EA)_2 = \left[\frac{\partial^2 \eta_2}{\partial t^2}(0, t)\right]m_1 \quad (11.3)$$

$$\eta_1(\ell_1, t) = \eta_2(0, t) \quad (11.4)$$

$$-\left[\frac{\partial \eta_2}{\partial z_2}(\ell_2, t)\right](EA)_2 = \left[\frac{\partial^2 \eta_2}{\partial t^2}(\ell_2, t)\right]m_2 \quad (11.5)$$

where the longitudinal wave velocities are given by:

$$c_{\eta_j} = \sqrt{(EA)_j/\mu_j} \quad j = 1, 2 \quad (12)$$

The displacement *FRF*'s $|\lambda_1|$ at the elevator and $|\lambda_2|$ at the upper-platform, and the attenuation function $|\theta_L|$ for the longitudinal waves are also obtained from equations (8) by performing the above mentioned transformation. We have therefore

$$|\lambda_1| = \left| \cos \gamma_1 - \omega b_1 \sin \gamma_1 + \xi \frac{\omega b_2 \sin \gamma_1}{\cos \gamma_2 - \omega b_2 \sin \gamma_2} \right|^{-1} \quad (13.1)$$

$$|\theta_L| = |\cos \gamma_2 - \omega b_2 \sin \gamma_2|^{-1} \quad (13.2)$$

$$|\lambda_2| = |\lambda_1| |\theta_L| \quad (13.3)$$

where

$$b_j = m_j c_{\eta_j} / (EA)_j \quad j = 1, 2 \quad (14.1)$$

$$y_j = \omega \ell_j / c_{\eta_j} \quad j = 1, 2 \quad (14.2)$$

$$\xi = \frac{(EA)_2 c_{\eta 1}}{(EA)_1 c_{\eta 2}} \quad (14.3)$$

The longitudinal displacements S_1 at the elevator and S_2 at the upper platform are

therefore given by

$$S_1(\ell_1) = p_L \lambda_1 \quad (15.1)$$

$$S_2(\ell_2) = p_L \lambda_1 \theta_L \quad (15.2)$$

The acceleration *FRF's* $|\lambda_1^a|$ and $|\lambda_2^a|$ are immediately obtained from equations (13.1) and (13.3) as

$$|\lambda_j^a| = \gamma_j^2 |\lambda_j| \quad j = 1, 2 \quad (16)$$

2.1.4 Numerical Results

The longitudinal and transverse-wave frequency response functions at the elevator and at the upper platform have been evaluated for the typical design parameters of the upper tether of *TECS*. Specifically (see Figure 1) we have: $(EA)_1 = (EA)_2 = 61645 \text{ N}$, $T_1 = 394.5 \text{ N}$, $T_2 = 375.5 \text{ N}$, and $\mu_1 = \mu_2 = 4.9 \times 10^{-3} \text{ kg/m}$. Consequently the transverse-wave velocities in tether segment 1 and 2 (of the upper tether) are

$$c_{\psi_1} = 283.74 \text{ m/s} \quad (17.1)$$

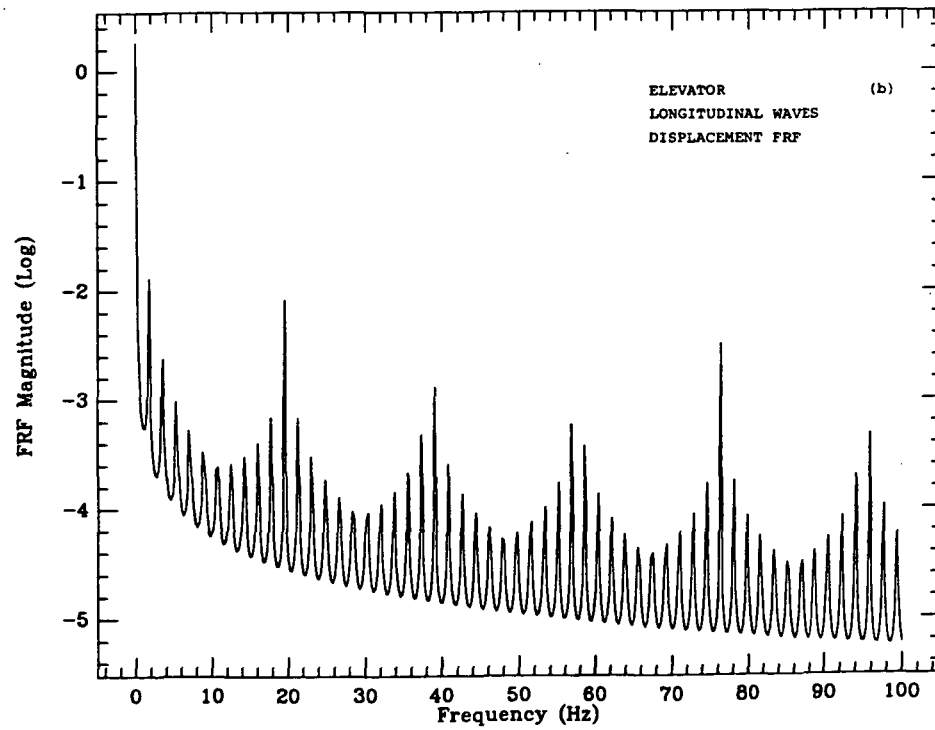
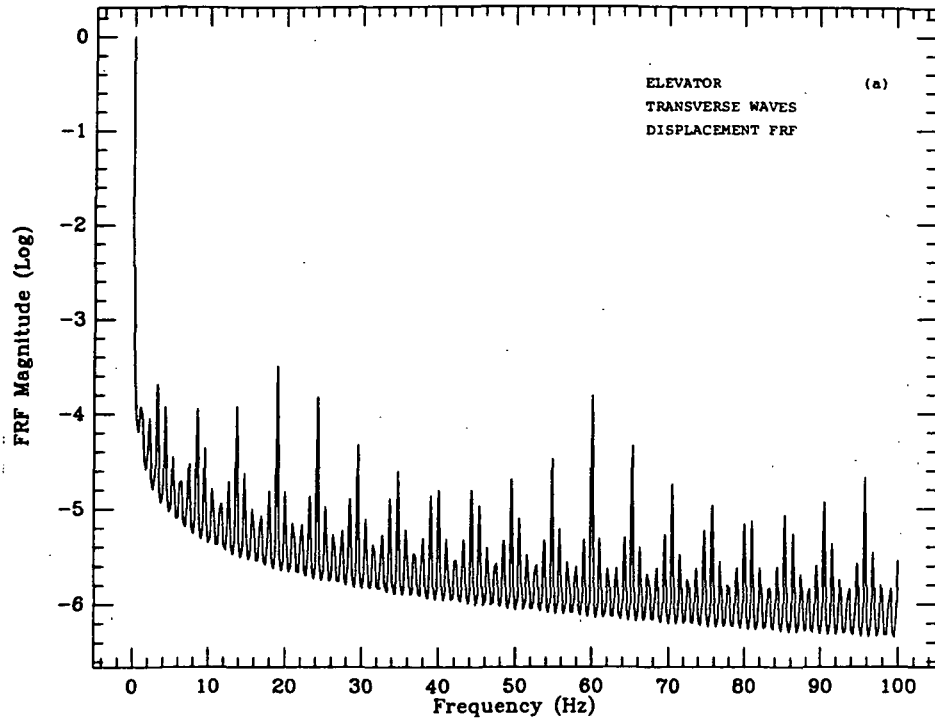
$$c_{\psi_2} = 276.83 \text{ m/s} \quad (17.2)$$

and the longitudinal-wave velocities

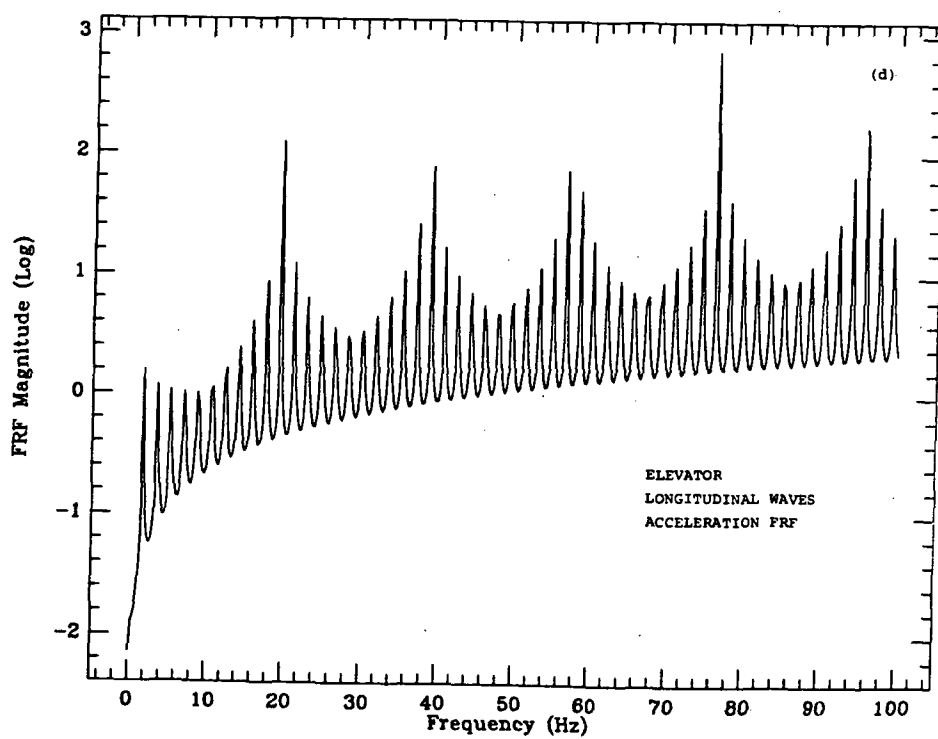
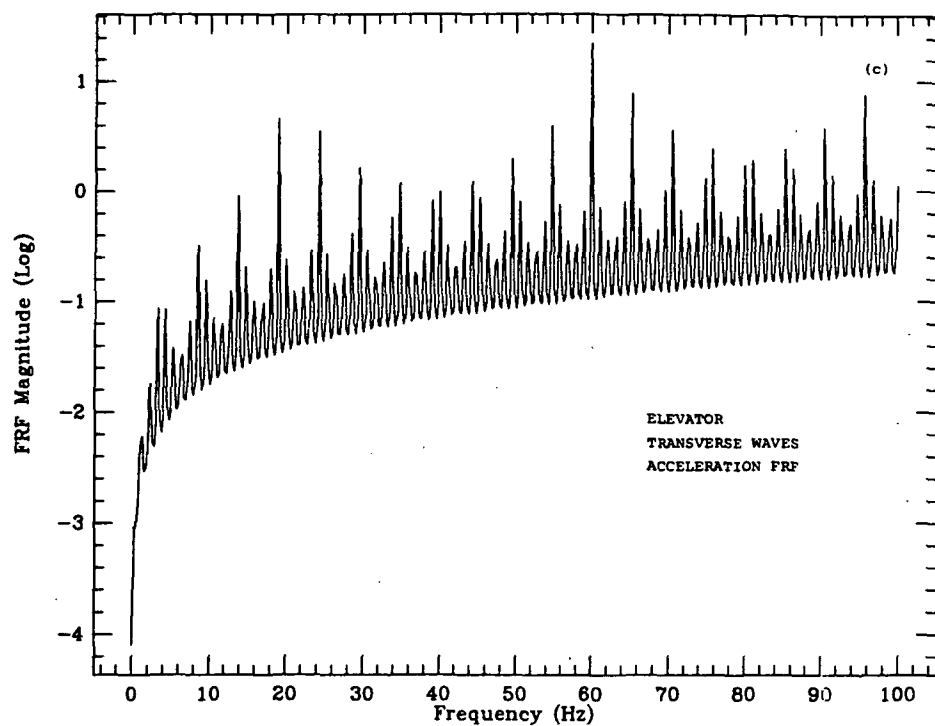
$$c_{\eta 1} = c_{\eta 2} = 3546.92 \text{ m/s} \quad (18)$$

The elevator displacement *FRF*'s $|\chi_1|$ (transverse waves) and $|\lambda_1|$ (longitudinal waves) are shown in Figures 2(a) and 2(b). Similarly the elevator acceleration *FRF*'s $|\chi_1^a|$ and $|\lambda_1^a|$ for the transverse and longitudinal waves are shown in Figures 2(c) and 2(d) respectively. The last two figures have been zoomed-in for frequencies less than 10 Hz in Figures 2(e) and 2(f). In general the tether segment behaves as the superposition of a single *DOF* low-pass filter and a transmission line for both longitudinal and transverse disturbances. In this analysis the effect of damping, both internal and external, has been neglected. In an actual case the resonance peaks will be limited by the damping. The lower envelopes of the displacement *FRF*'s correspond to the response of a single *DOF* low-pass filter while the additional spectral lines are related to the transverse and longitudinal waves propagating in the tether.

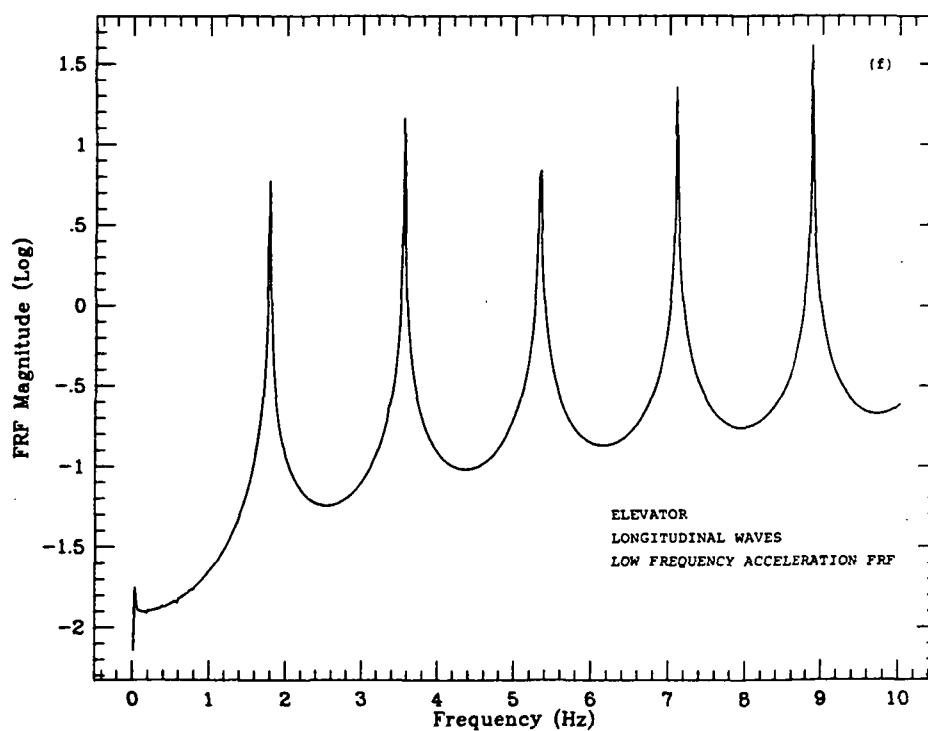
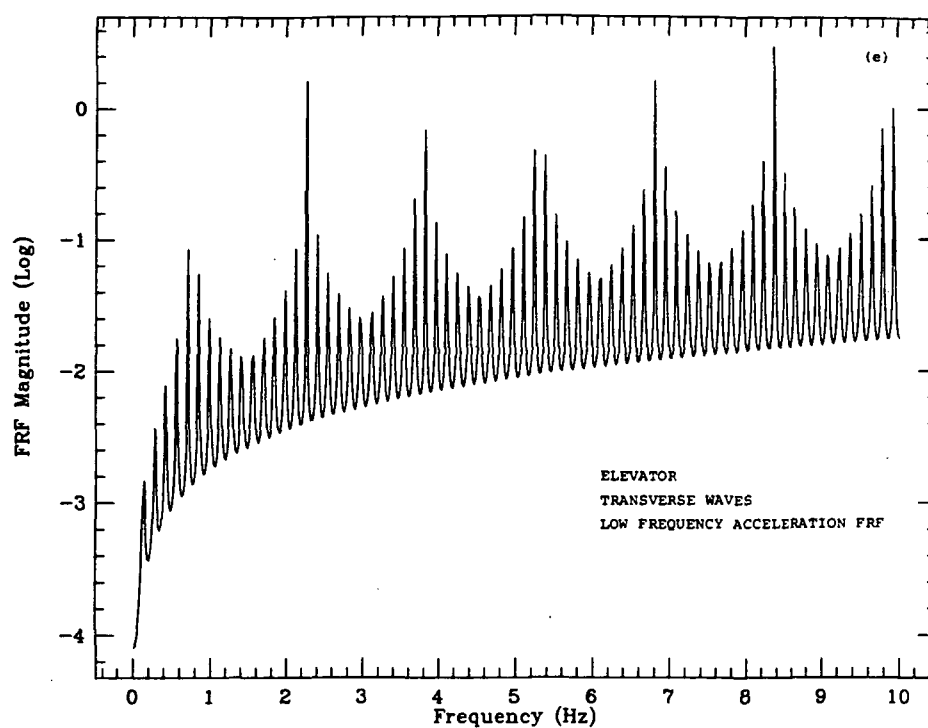
Figures 3(a)-3(f) are like Figures 2(a)-2(f) except that the former set is referred to the upper platform. By comparing these two sets of figures it is clear that the elevator acts as a wave attenuator. The noise at the upper platform is a few orders of magnitude smaller than at the elevator. Specifically Figures 3(g) and 3(h) show the displacement attenuation functions $|\theta_T|$ and $|\theta_L|$ for transverse and longitudinal waves respectively. The acceleration attenuation



Figures 2(a) and 2(b)



Figures 2(c) and 2(d)



Figures 2(e) and 2(f)

functions $|\theta_T^a|$ and $|\theta_L^a|$ are readily obtained as follows:

$$|\theta_T^a| = (\phi_2/\phi_1)^2 |\theta_T| \quad (19.1)$$

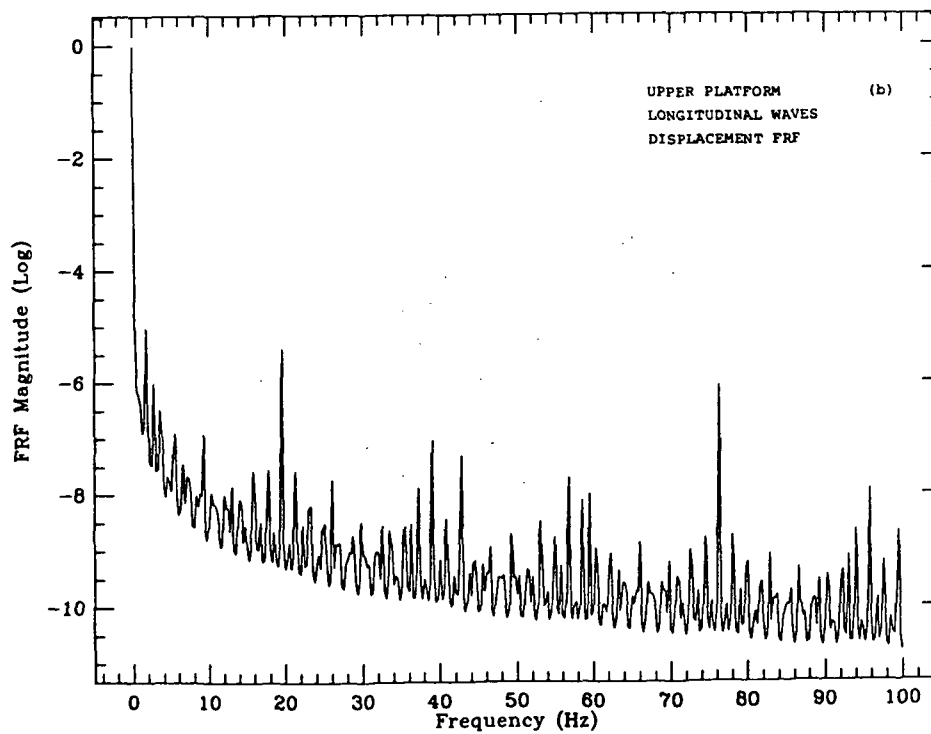
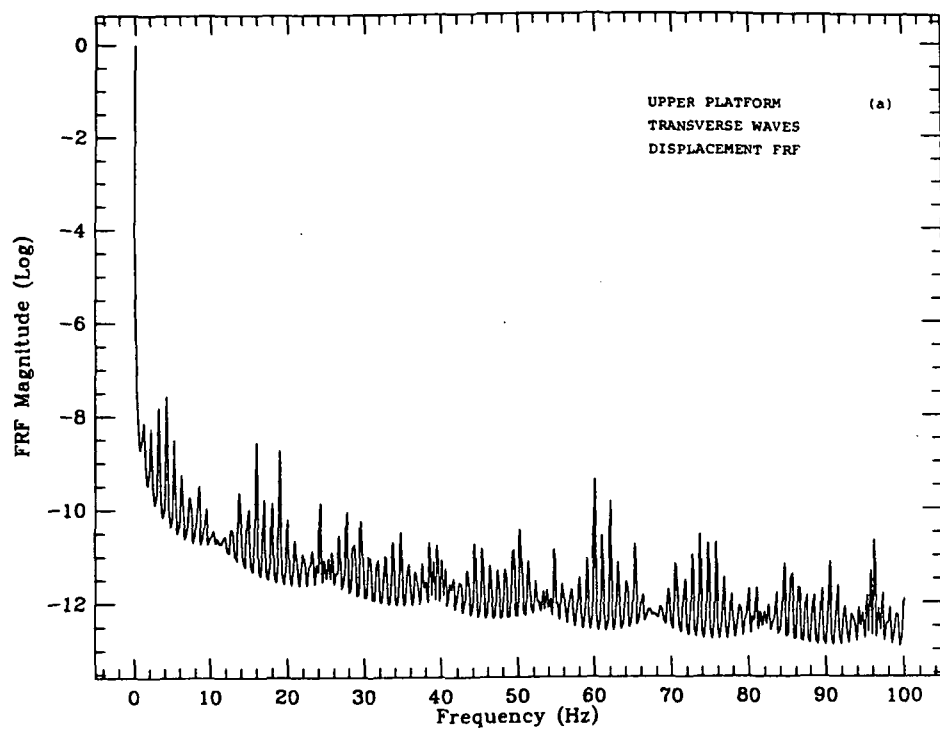
$$|\theta_L^a| = (\gamma_2/\gamma_1)^2 |\theta_L| \quad (19.2)$$

Figures 3(g) and 3(h), therefore, provide also the acceleration attenuation functions when their magnitudes are scaled by

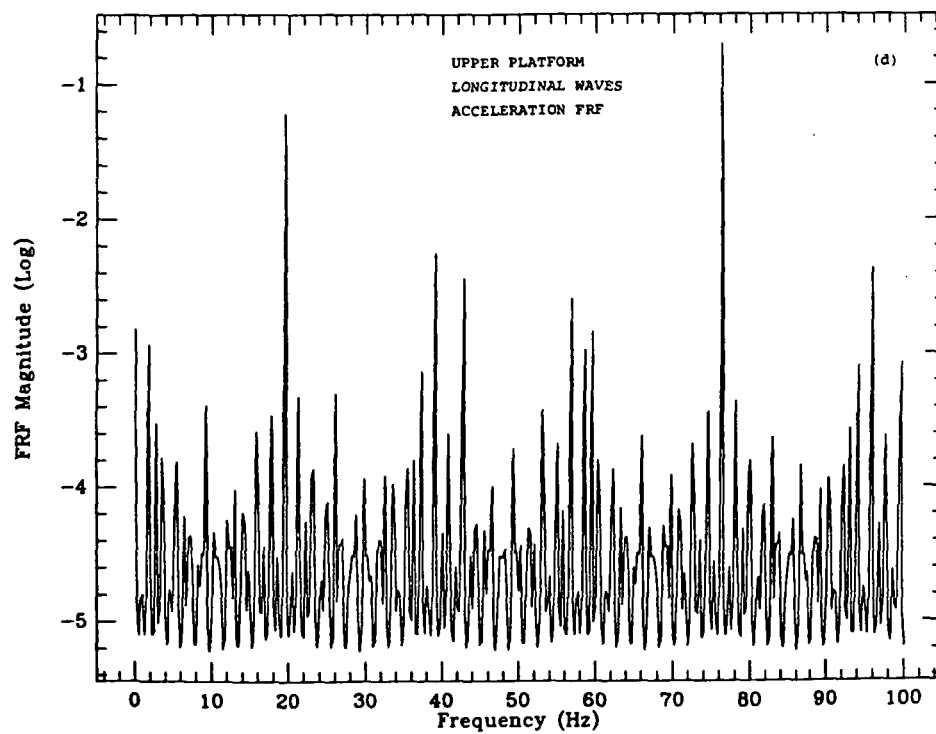
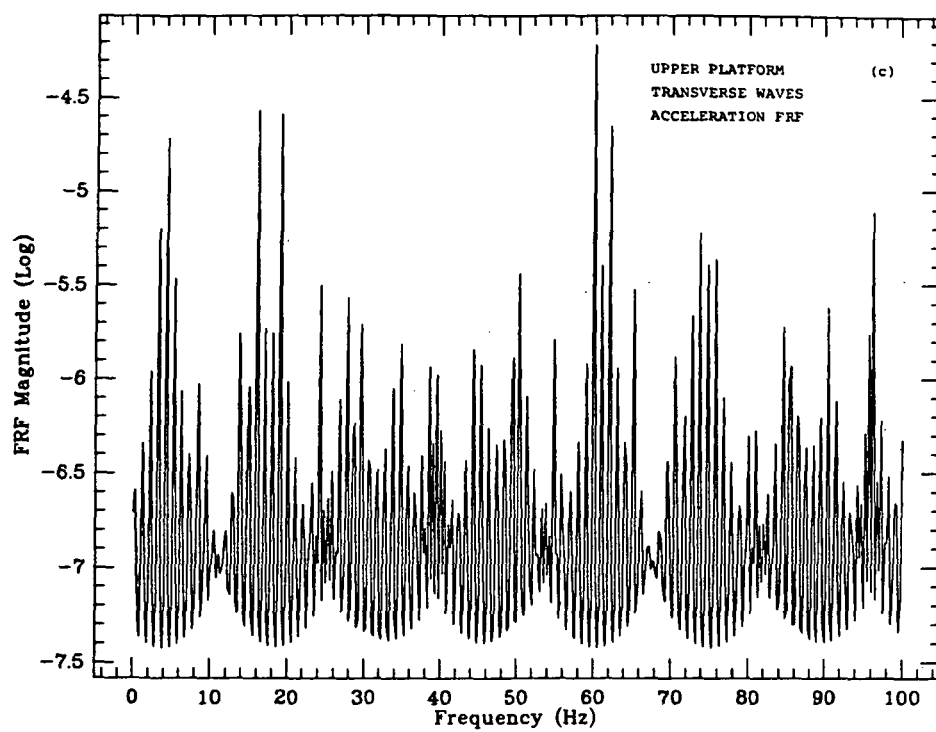
$$(\phi_2/\phi_1)^2 = \left(\frac{\ell_2 c_{\psi 1}}{\ell_1 c_{\psi 2}} \right)^2 = 85 \quad (20.1)$$

$$(\gamma_2/\gamma_1)^2 = \left(\frac{\ell_2 c_{\eta 1}}{\ell_1 c_{\eta 2}} \right)^2 = 81 \quad (20.2)$$

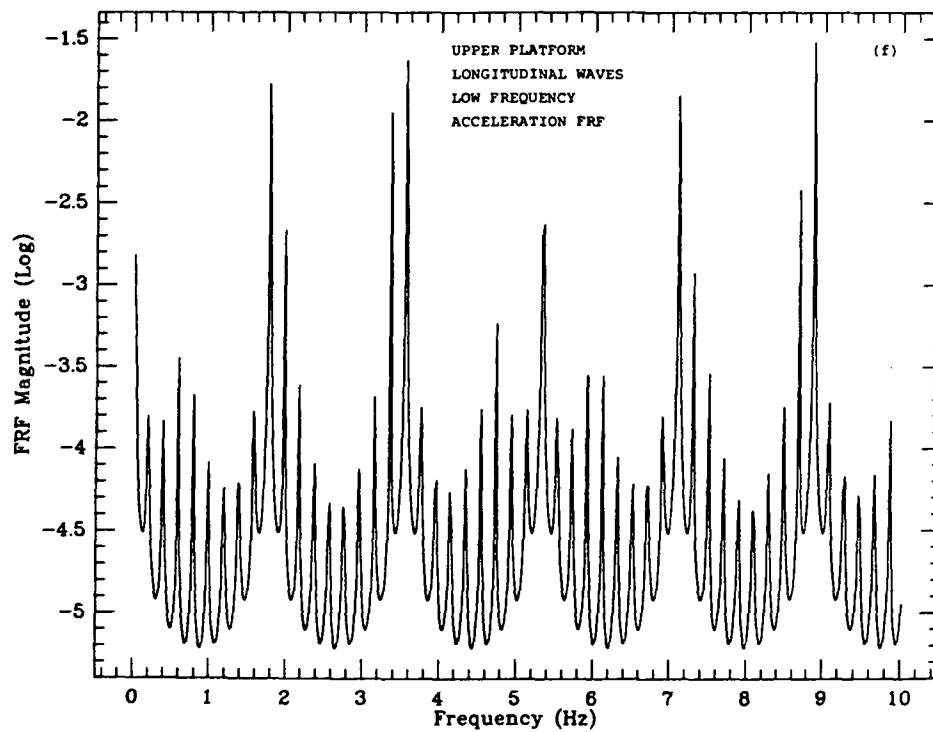
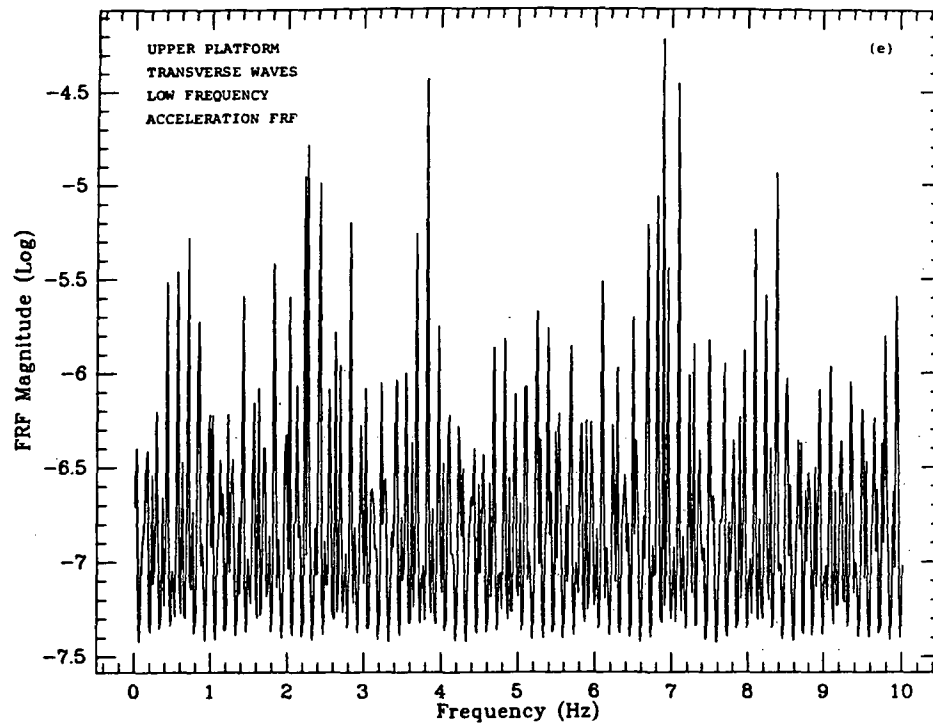
The response at the upper platform stresses the previous conclusion: the upper tether acts as the superposition of a low-pass filter and a transmission line. The elevator, however, attenuates the transverse and the longitudinal waves propagating from tether segment 1 (below the elevator) to tether segment 2 (above the elevator).



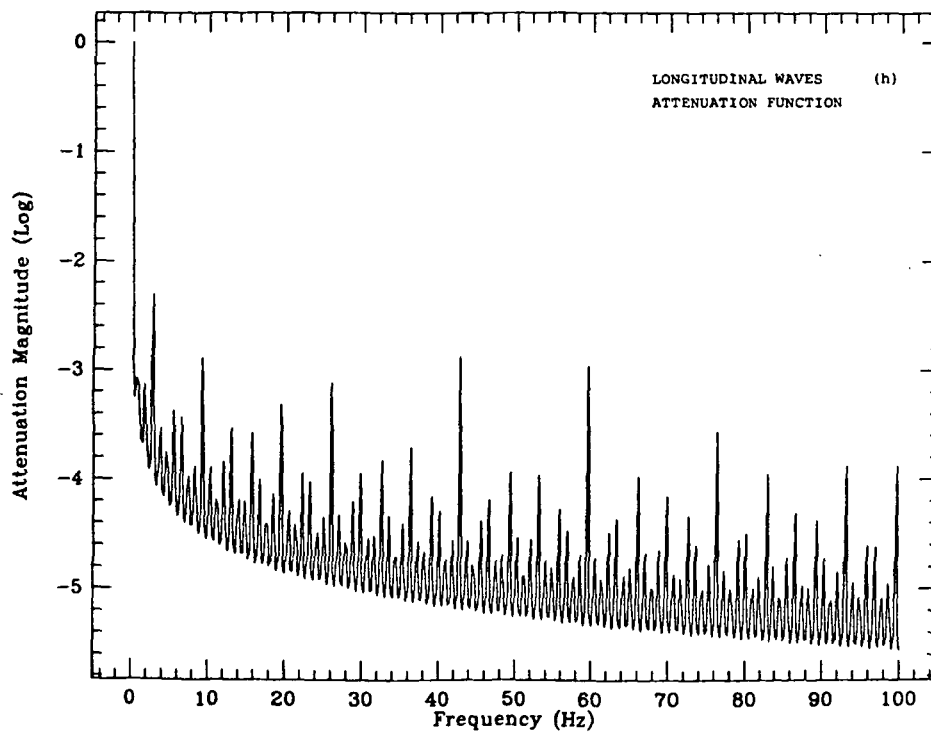
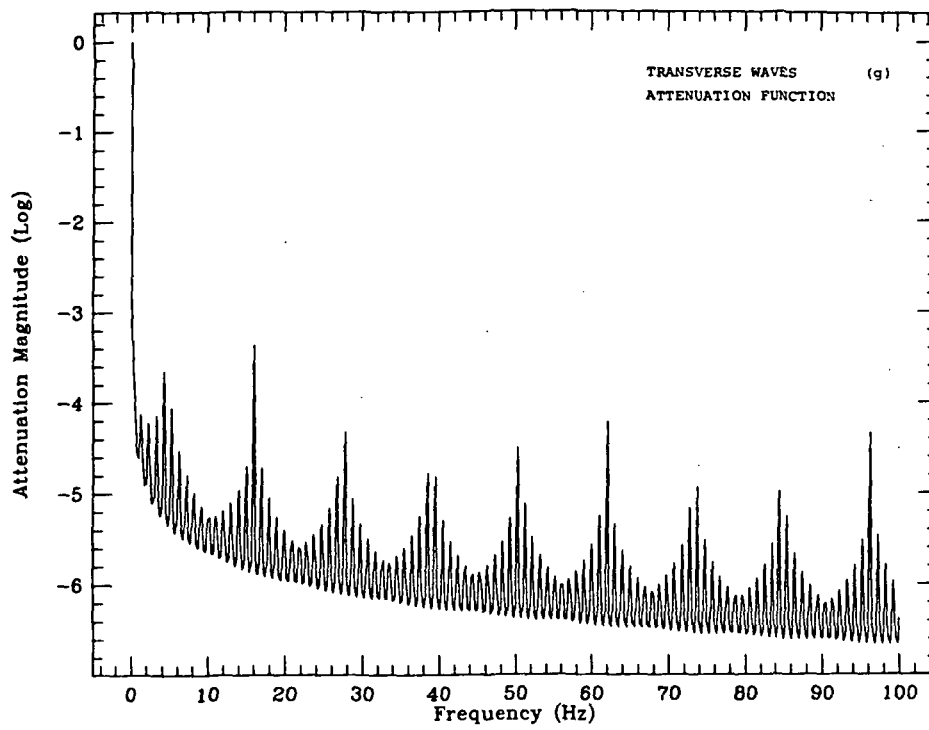
Figures 3(a) and 3(b)



Figures 3(c) and 3(d)



Figures 3(e) and 3(f)



Figures 3(g) and 3(h)

2.1.5 References To Section 2.1

1. Xiaohua He, and J.D. Powell, "Tether Damping in Space," to appear in Proceedings of the Second International Conference on Tethers in Space, Venice, Italy, 4-8 October 1987.
2. G.E. Gullahorn and R.G. Hohlfeld, "Tether as a Dynamic Transmission Line," to appear in Proceedings of the Second International Conference on Tethers in Space, Venice, Italy, 4-8 October 1987.

2.2 Longitudinal Dampers

2.2.1 Introductory Remarks

The damping of longitudinal (along the tether segments) oscillations is essential to provide acceleration levels on board the elevator and the station suitable for microgravity experiments [1]. Specifically once the deployment-induced-oscillations have been dissipated the libration and lateral oscillation control systems can be switched off without impairing the microgravity experiments [1]. In fact the steady-state external perturbations are not strong enough to excite appreciably the libration and the lateral oscillations. On the contrary, thermal perturbations have a non-negligible effect upon the longitudinal oscillations. Everytime the system crosses the terminator, the tethers' temperatures vary abruptly (almost impulsively) and longitudinal oscillations are excited twice per orbit. Longitudinal oscillations, therefore, must be continuously controlled if low acceleration levels are desired.

In reference [2] we have described the three longitudinal dampers which are in series with the three tether segments connecting massive platforms. In reference [2] we also selected each damper parameters (stiffness and damping coefficient) based on considerations arising from physical intuition. Each damper has been tuned to the frequency of the natural bobbing frequency of the associated tether segment. We also adopted a damping coefficient which provides a damping

ratio equal to 0.9 for an ideal damper which is decoupled from the associated tether (i.e. the ideal damper is directly connected to the platform). This selection of parameters has indeed provided effective damping of tether bobbing oscillations. Nevertheless, since the performance of the longitudinal dampers is essential to guarantee a microgravity environment on board the tethered system, we want to verify more rigorously the validity of our selection of the damper parameters. We also want to assess if a detuned damper (i.e. damper stiffness different from the tether stiffness) can possibly perform better than a tuned longitudinal damper.

2.2.2 Optimization Of Longitudinal Damper Parameters

A relatively accurate mathematical model, amenable to an analytical solution, for each damper and the associated vibrating tether segment is as follows: each damper is assumed massless and only two degrees of freedom (DOF) are taken into account, i.e. the damper stretch and the elastic stretch of the associated tether segment (see Figure 4). The assumption of massless damper is actually quite accurate for an active control system. As a result of high feedback gains, the active damper's response exhibits a negligible phase delay and therefore the effects of the damper's (spool's) inertia are masked. The two-DOF-assumption provides a realistic picture of the actual behavior of the system for small values of lateral oscillations since the coupling between the lateral and the longitudinal oscillations decreases quadratically with the lateral oscillation amplitude. This case

of “quasi”-aligned tethered system is actually the most interesting because is representative of the system steady-state configuration.

With reference to Figure 4 the force-balance equations for the longitudinal vibrations of one tether segment with damper are as follows:

$$m_Q \ddot{L} - F = F_E \quad (21.1)$$

$$F = F_d \quad (21.2)$$

where the equivalent mass m_Q is equal to $m_1 \cdot m_2 / (m_1 + m_2)$, L is the geometrical distance between two adjacent platforms with masses m_1 and m_2 , F_E is an arbitrary external force. The forces F (tether) and F_d (damper) are given by:

$$F = -k(L - \ell_o - \ell_d) = -k\ell_t \quad (22.1)$$

$$F_d = -\left(k_d \ell_d + b \dot{\ell}_d\right) \quad (22.2)$$

where ℓ_o is the unstretched tether length, ℓ_d is the damper stretch, k is the tether stiffness, k_d and b are the stiffness and damping coefficient of the damper, and ℓ_t is the elastic stretch of the tether.

After substitution of equations (22) into equations (21) and after some algebraic manipulations, we obtain the equations of motion as follows:

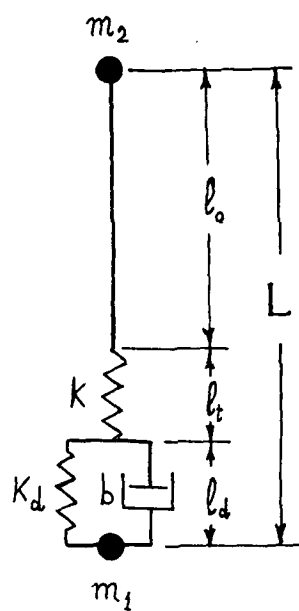


figure 4

$$\ddot{\ell} + \omega_o^2 \ell - \omega_o^2 \ell_d = F_E/m_Q \quad (23.1)$$

$$\dot{\ell}_d + 2\chi_d \ell_d - \chi \ell = 0 \quad (23.2)$$

where $\ell = L - \ell_o$ is the total stretch of the vibrating system. We have also defined

$$\begin{aligned} \omega_o^2 &= k/m_Q \\ \chi_d &= k_d/b \\ \chi &= k/b \end{aligned} \quad (24)$$

Equations (23) describe the motion of a 3rd-order vibrating system which is actually amenable to an analytic solution.

By Laplace transforming equations (23), with null initial conditions, we obtain

$$s^2[\ell] + \omega_o^2[\ell] - \omega_o^2[\ell_d] = [F_E]/m_Q \quad (25.1)$$

$$s[\ell_d] + 2\chi_d[\ell_d] - \chi[\ell] = 0 \quad (25.2)$$

where $[]$ denotes the Laplace transformation. After solving equation (25.2) for $[\ell_d]$ and substituting into equation (25.1) we obtain

$$[\ell_d] = \frac{\chi}{s + 2\chi_d} [\ell] \quad (26.1)$$

$$[\ell] \left\{ s^2 + \omega_o^2 - \omega_o^2 \chi / (s + 2\chi_d) \right\} = [F_E] / m_Q \quad (26.2)$$

which after substitution into equation (25.1) provides the characteristic equation of the system

$$s^3 + 2\chi_d s^2 + \omega_o^2(2\chi_d - \chi) = 0 \quad (27)$$

By using the transformation $s = z - 2\chi_d/3$ equation (27) reduces to the canonical form

$$z^3 + pz + q = 0 \quad (28)$$

where

$$\begin{aligned} p &= \omega_o^2 - \frac{4}{3}\chi_d^2 \\ q &= \omega_o^2 \left(\frac{4}{3}\chi_d - \chi \right) - \frac{8}{27} \chi_d^3 \end{aligned} \quad (29)$$

The discriminant of the cubic equation (28) is

$$\Delta = \left(\frac{q}{2} \right)^2 + \left(\frac{p}{3} \right)^3 \quad (30)$$

For an oscillatory response of the system we must have $\Delta \geq 0$ which implies that a pair of roots are complex conjugate and one root is real. For $\Delta < 0$ the roots are all real. Real roots of equation (28) implies real roots of equation (29). By applying the Routh criterion to equation (29), however, we discover that at least one of the three real roots is positive. Consequently the system is unstable for $\Delta < 0$ and we limit our parametric analysis to the case $\Delta \geq 0$.

Instead of using Cardano's formulae (which are quite cumbersome) for computing the roots of the cubic equation (27), we compute the real root s_1 by means of a numerical root finding routine and then we deflate equation (27). After deflation, equation (27) becomes

$$(s - s_1)[s^2 + (s_2 + 2\chi_d)s + (s_1 + 2\chi_d)s_1 + c] = 0 \quad (31)$$

Since equation (31) has real coefficients, the roots s_2 and s_3 of the 2nd-order equation between square brackets are complex conjugate. After defining

$$\begin{aligned} s_2 &= \beta + i\omega \\ s_3 &= \beta - i\omega \end{aligned} \quad (32)$$

we can express the characteristic equation (27) as

$$D(s) = (s - s_1)[s - (\beta + i\omega)][s - (\beta - i\omega)] \quad (33)$$

From the solution of the quadratic equation between square brackets in equation (31) we have

$$\begin{aligned}\beta &= -(s_1 + 2\chi_d)/2 \\ \omega &= \frac{1}{2} \left\{ (s_1 + 2\chi_d)^2 - 4[(s_1 + 2\chi_d)s_1 + \omega_o^2] \right\}^{1/2}\end{aligned}\quad (34)$$

We can express $[\ell]$ and $[\ell_d]$ in terms of transfer functions by using equations (26)

$$[\ell] = G_1(s)[F_E]/m_Q \quad (35.1)$$

$$[\ell_d] = G_2(s)[F_E]/m_Q \quad (35.2)$$

The transfer functions are given by

$$G_1(s) = (s + 2\chi_d)/D(s) \quad (36.1)$$

$$G_2(s) = \chi/D(s) \quad (36.2)$$

where $D(s)$ is the characteristic equation.

The expressions of ℓ and ℓ_d as a function of time [i.e. the solution of the 3rd-order differential equations (23)] are obtained by Laplace reverse-transforming equations (35). A case of particular theoretical and practical interest is the

response of the system to an impulse. If we assume that F_E is an impulsive function of strength I we have that

$$[F_E] = I = \text{constant} \quad (37)$$

and equations (35) can be Laplace reverse-transformed by a lengthy but well known technique. $G_1(s)$ and $G(s)$ are first reduced to a sum of simple fractions as follows

$$G_1(s) = \frac{A_1}{s - s_1} + \frac{B_1 s}{D_2(s)} + \frac{B_2}{D_2(s)} \quad (38.1)$$

$$G_2(s) = \frac{\bar{A}_1}{s - s_1} + \frac{\bar{B}_1 s}{D_2(s)} + \frac{\bar{B}_2}{D_2(s)} \quad (38.2)$$

where $D_2(s) = [s - (\beta + i\omega)][s - (\beta - i\omega)]$ and the coefficients A_1 , B_1 , B_2 , \bar{A}_1 , \bar{B}_1 , and \bar{B}_2 are functions of χ_d, χ and ω_0 .

The variable of greatest interest to us is actually the elastic stretch

$$\ell_t = \ell - \ell_d \quad (39)$$

Hence, by taking into account that the Laplace transform is a linear operator, we can define a third transfer function $G_3(s)$ as follows:

$$[\ell_t] = G_3(s)I/m_Q \quad (40.1)$$

where

$$G_3(s) = G_1(s) - G_2(s) \quad (40.2)$$

Subtracting equation (38.1) from equation (38.2) we obtain the fraction formulation of $G_3(s)$ and we can finally Laplace reverse-transform equation (40.1). We give only the final result, without providing details on the reverse-transformation for the sake of brevity.

The elastic stretch as a function of time for null initial conditions and impulsive external force is given by

$$\ell_t = \frac{I}{m_Q} \left\{ A e^{s_1 t} - e^{\beta t} [A \cos(\omega t) + B \sin(\omega t)] \right\} \quad (41)$$

where s_1 , β and ω have already been defined and

$$A = \frac{2\chi_d - \chi + s_1}{3s_1^2 + \omega_o^2 + 4\chi_d s_1} \quad (42)$$

$$B = \frac{2A(\chi_d + s_1) - 1}{\omega}$$

Equation (41) gives the response of the elastic stretch as a function of the

parameters of the 3rd-order vibrating system χ , χ_d , and ω_n .

Since the acceleration measured on any platform of the tethered system is directly proportional to the tether elastic stretch, the time history of the latter quantity is of particular significance to the microgravity levels of our tethered system. In an actual case a tethered system is excited twice per orbit by a “quasi” impulsive thermal perturbation which deposits energy primarily into the longitudinal oscillations. Our goal is therefore to define those values of parameters χ , χ_d and ω_n which provides the smallest and shortest fluctuation of the acceleration.

Several parameters can be used as indicators of the effectiveness of the damper. We adopt the settle time t_s and the maximum elastic stretch per unit initial velocity $\bar{\ell}_t$ [in equation (40.1) I/m_Q is equal to the initial velocity] as primary indicators of the damper’s effectiveness. We define the settle time as the time taken by the system to reduce the oscillation amplitude to 10% of its maximum value after the application of the external impulse. We also define the rise time t_M as the time to reach the maximum amplitude of the response and the relaxation time t_R as the time the system takes to reduce the amplitude from its maximum value to 33% of its maximum amplitude (i.e. $\bar{\ell}_t/e$).

The rise time and the maximum elastic stretch have been computed for several values of χ and χ_d by bracketing equation (40.1) around its absolute maximum and then using a root-finding routine to locate the maximum value with

high accuracy. The computation of the settle time and relaxation time require knowledge of the envelope of ℓ_t . Such an envelope is given by:

$$\hat{\ell}_t = \frac{I}{m_Q} \left(|A| e^{s_1 t} + |M| e^{\beta t} \right) \quad (43)$$

where $M = \left(A^2 + B^2 \right)^{1/2}$. The relaxation time is then computed by solving numerically the following equation

$$|A| e^{s_1(t_R + t_M)} + |M| e^{\beta(t_R + t_M)} = \bar{\ell}_t / e \quad (44)$$

where t_M is the previously computed rise time. The settle time t_S is also computed by equation (44) after replacing the factor e with 10. We want to point out that in general $t_S > t_M + t_R$.

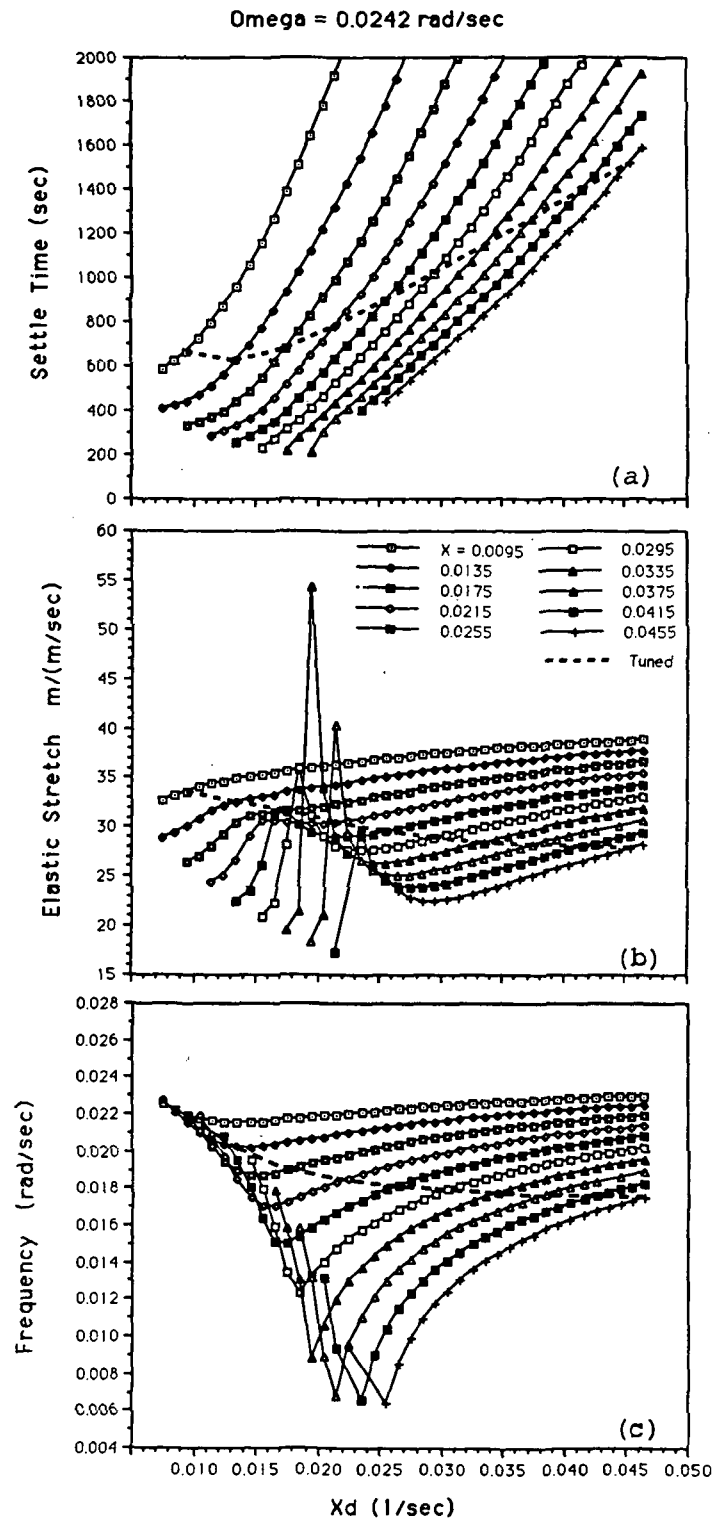
Figures 5(a)-5(c) depict the settle time, the maximum elastic stretch per unit initial velocity, and the pseudo frequency vs. χ and χ_d for $\omega_n = 0.0242$ rad/sec ($f_n = 3.844 \times 10^{-3}$ Hz). This value of ω_n is the natural bobbing frequency of the first tether segment (i.e. between the lower mass and the station).

The settle time, in Figure 5(a), depends almost linearly upon the damper parameter χ_d . The dashed line connects the points with $\chi = \chi_d$ which are representative of a damper tuned to the bobbing frequency of the associated tether

segment. The shortest settle time for a tuned damper is obtained for $\chi = \chi_d = 0.0135 \text{ sec}^{-1}$ which is actually the value adopted in reference [6] for the longitudinal damper of tether 1. The settle time for a detuned damper can be shorter than that for a tuned damper as shown by all the points below the dashed line. From inspection of Figure 5(b) we notice, however, that some of the above mentioned points (for a detuned damper) correspond to maximum elastic stretches much larger than those associated with a tuned damper. There are only small areas in the $\chi - \chi_d$ plane which correspond to a short settle time and a small elastic stretch for a detuned damper. One of such areas is for those values of χ and χ_d along the lowest points of Figures 5(a) and 5(b). Below those points the response is unstable and a certain margin should be provided to account for possible fluctuations of χ and χ_d .

Figure 5(c) shows the pseudo-frequency ω as a function of χ and χ_d for $\omega_o = 0.0242 \text{ rad/sec}$. It is worth noticing the reduction of the pseudo-frequency for certain values of χ and χ_d of a detuned damper.

Figures 6(a)-6(c) depict the settle time, the elastic stretch, and the pseudo frequency respectively vs. χ and χ_d for $\omega_o = 0.117 \text{ rad/sec}$ ($f_o = 1.868 \times 10^{-2} \text{ Hz}$) which is the natural bobbing frequency of tether segment 2 (between the station and a 1-km-off elevator). The trends of these plots are similar to the previous set of plots except for the ranges of the parameters χ and χ_d which are larger than before. There may be a scaling factor related to ω_o even if it is not immediately



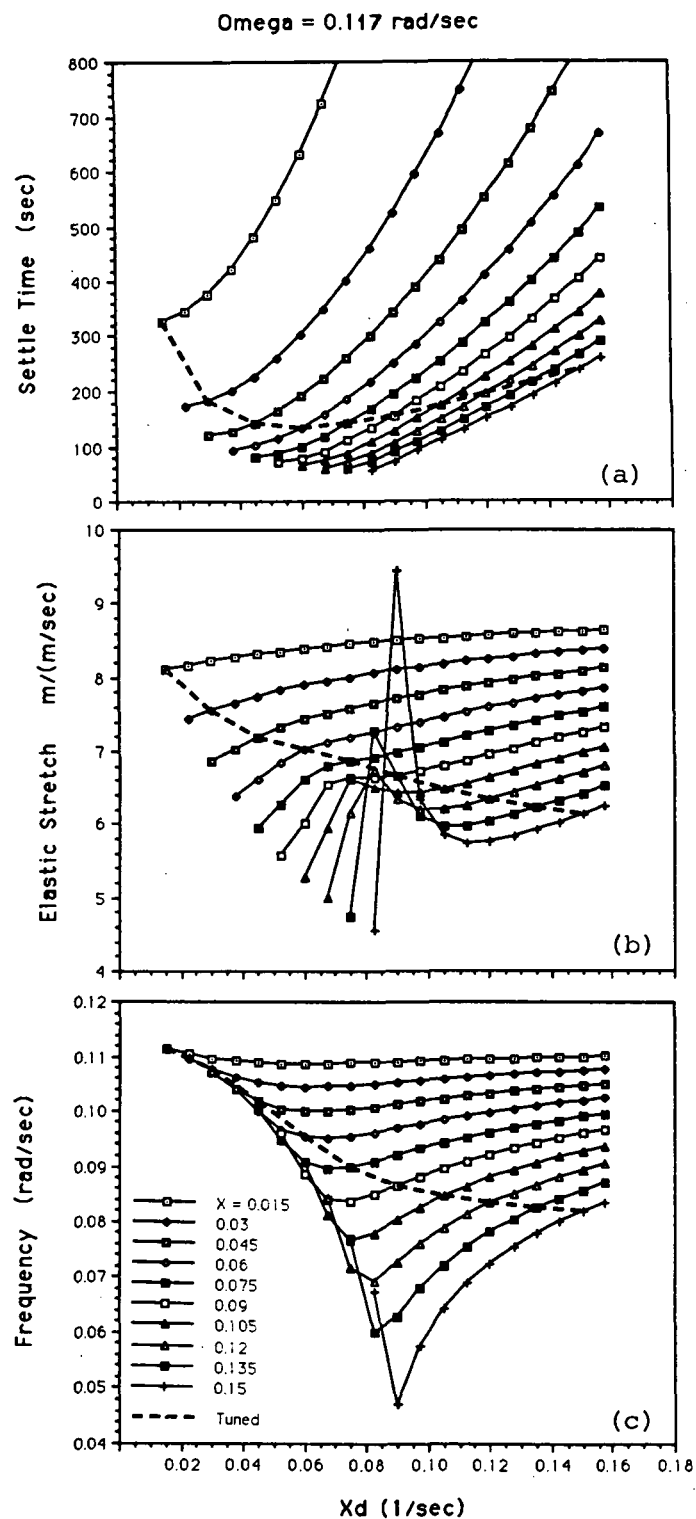
Figures 5(a)-5(c)

evident from the inspection of the equations.

This time the shortest settle time for a tuned damper is even closer to the best results obtained by adopting a detuned damper. The detuned damper is also affected, for certain values of the parameters, by the problem of the amplification of the elastic stretch. We prove again that the values adopted in reference [6] for the longitudinal damper of tether 2, $\chi = \chi_d = 0.065 \text{ sec}^{-1}$, provides the shortest settle time for a tuned damper. The figures for tether segment 3 are not shown because they are quite similar to Figures 5(a)-5(c) of tether segment 1. The natural frequency of tether segment 3 is $\omega_o = 0.0247 \text{ rad/sec}$ ($f_o = 3.934 \times 10^{-3} \text{ Hz}$) which is very close to the natural frequency of tether segment 1. In conclusion, because of the moderate advantages of a detuned damper over a tuned damper we have decided to adopt tuned longitudinal dampers, as in reference [1], with the following values of parameters

$$\begin{aligned}\chi_1 &= \chi_{d1} = 0.0135 \text{ sec}^{-1} \\ \chi_2 &= \chi_{d2} = 0.065 \text{ sec}^{-1} \\ \chi_3 &= \chi_{d3} = 0.0137 \text{ sec}^{-1}\end{aligned}\tag{45}$$

The indexes in equations (45) refer to the three tether segments respectively.



Figures 6(a)-6(c)

2.2.3 References To Section 2.2

1. E.C. Lorenzini, M. Cosmo, S. Vetrella and A. Moccia, "Acceleration Levels on Board the Space Station and a Tethered Elevator for Micro and Variable-Gravity Applications," to appear in Proceedings of 2nd-International Conference on Tethers in Space," Venice, Italy, 4-8 October 1987.
2. E.C. Lorenzini et al., "Analytical Investigation of the Dynamics of Tethered Constellations in Earth Orbit (Phase II)," SAO Quarterly Report #2 under contract NASA/MSFC NAS8-36606.

2.3 Concluding Remarks

A perturbation originated from the space station produces frequency responses at the elevator and at the upper-platform which contain frequency components related to the longitudinal and transverse waves propagating in the upper-tether. These frequency responses can be viewed as the superposition of the responses of a single-degree-of-freedom low-pass filter and of a transmission line. This result is not discouraging because the higher frequency longitudinal waves could be abated by the longitudinal dampers which are tuned to the low frequency bobbing frequencies. The abatement of transverse waves, on the other hand, requires a low-frequency transverse wave attenuator. By analyzing the results of our analysis we notice that the elevator attenuates strongly both the longitudinal and transverse waves travelling from the tether segment below the elevator to the

one above. This result provides an indication of how to design a passive wave-attenuator for transverse waves which will be investigated during the next reporting period.

The wave propagation analysis stresses once more the importance of the longitudinal dampers not only as devices for damping out the bobbing oscillations but also for attenuating the longitudinal waves. We have, therefore, carried out an analysis to optimize the damping performance of the longitudinal dampers. The results of this analysis shows that detuned dampers may be actually even more effective than tuned dampers in abating longitudinal oscillations. The advantages are, however, quite limited. Our analysis has confirmed that the damper parameters, that we had selected for the dampers, are "optimal" with respect to the damping time for tuned dampers. Since the bobbing frequencies are the lowest frequencies of the longitudinal wave frequency-response-functions the longitudinal dampers are also expected to act as low-pass filters. These issues related to the attenuation of longitudinal and transverse waves will be investigated in the next reporting period.

3.0 PROBLEMS ENCOUNTERED DURING REPORT PERIOD, E.C. LORENZINI, PI

None

4.0 ACTIVITY PLANNED FOR NEXT REPORTING PERIOD, E.C. LORENZINI, PI

In the next reporting period we will analyze the techniques for attenuating the longitudinal and transverse wave propagating from the space station to the upper-platform along the upper tether.

We will also work on the implementation of the rotational dynamics of the station and the elevator into *MASTER20* computer code.

5.0 TECHNICAL ACTIVITY DURING REPORTING PERIOD AND PROGRAM STATUS, G. GULLAHORN PI.

A variety of other demands (response to time critical demands of the TSS1 mission as PI for an experiment on board, startup of a new grant, and internal administrative duties) reduced the effort the PI could devote to these tasks. Primary activity revolved around acquisition and setup of a new microcomputer system and transporting the SLACK code to that system. It is expected that greater effort can be devoted in the next reporting period.

5.1 Tether Applications Simulation Working Group Support

No significant activity in this reporting period.

5.2 Tether Aerodynamic Effect Of RCS Thruster Plume

The activity during the reporting period involved acquisition and setup of a microcomputer system and transfer of Fortran code for the SLACK program to the new system.

5.2.1 Computer Acquisition

A Northgate microcomputer system with a 12 MHz 80286 central processor and 8 MHz 80287 numeric coprocessor was acquired. The system includes 1 MB of system memory, a 60 MB hard disk, and a monochrome graphics display. A variety of software was either transferred from a home computer system or may be used on both systems under manufacturer's licensing agreements: compilers for Fortran, Pascal and C; wordprocessors; math libraries. Some further utility software was purchased: Pascal debugger; multitasking and task switching utilities; an 8087 library for the Fortran.

It should be noted that an independently clocked 12 MHz 80287 (Microway) can be obtained at reasonable cost which would increase floating point throughput by up to 50%. We have also requested institutional (overhead) funds to upgrade the basic system to an 80386/80387 system, which should about double the system throughput initially, and with proper compilers (taking advantage of instructions and registers unique to the 386/387) increase floating point throughput by factors

of 4 to 10.

5.2.2 Transport Of SLACK Code To Microcomputer

Physical transfer of the source code for the SLACK program involved little difficulty; indeed, it was already available on diskette, having been transferred via modem for distribution to those requesting the source.

Compilation and linking of the program proved not so simple. When written on the VAX full use had been made of a number of convenient extensions to the Fortran-77 standard provided in the VAX compiler (and one non-standard feature which though convenient is not simply an extension and which can lead to subtle errors). The compiler available (Microsoft) on the microsystem has few of these extensions, and a review of other microcomputer Fortrans showed that none of them supported the full set of extensions used. Hence a program FILTER was written (about 400 lines of Turbo Pascal) which accepts as input a VAX Fortran subroutine and outputs a version much closer to the F77 standard, flagging some constructs for manual modification. FILTER does the following

- Removes in-line comments to a separate line. VAX Fortran allows a comment to be placed on the same line with code, set off after an exclamation point. F77 requires all comments to be on a separate line with a C in the first column.
- Converts continuation lines starting with TAB to begin with 5 spaces.
- Convert continuation line with continuation character '0' to continuation character '1'. Microsoft does not allow '0'.

- Convert initial TAB to 7 spaces. Microsoft would accept these lines, but this is done for consistency with the above.
- Convert DO / ENDDO pairs to statement label format. VAX Fortran allows an extension in which the end of the range of a DO statement is delimited by an 'ENDDO', while F77 requires a labeled statement (referred to in the DO statement itself) as the delimiter. A statement label is provided (a five digit number), inserted in the DO statement, and attached to a created CONTINUE statement at the end of the range.
- Flag DO WHILE statements for manual modification. This useful extension repeats a loop while some condition is met (e.g., DO WHILE (X > 1.0)). This construct is replaced by a conditional branch around the loop and an unconditional branch at the end. It proved easier to simply flag these and modify them by hand.
- Flag DATA and COMMONS statements for possible manual (a) transfer to BLOCK DATA (b) change of the commons name if same as subroutine name. VAX Fortran allows a labeled commons to have the same name as a subroutine; F77 requires all program units (including commons and subroutines) to have distinct names. VAX Fortran allows initializing of variables in commons with a DATA statement. F77 requires that this be done in a separate BLOCK DATA subprogram, and on reflection the VAX permissibility, though convenient, can lead to subtle bugs: if data statements are used to initialize the same variable in more than one subroutine, tests show that the linker does not pick up this inconsistency, and the value assigned depends on the order in which routines are linked. Good programming practice would: initialize variables local to a subroutine with data statements and not put them in commons, as this is not needed in the F77 standard (the habit of putting local variables in commons when their value must be retained between subroutine calls was developed using a non-standard, stack-oriented compiler); use parameter statement initialization where values need not be modified; use block data or other distinct initialization (as in a startup routine) where variables must be referred to by several routines.

The SLACK code was run through the filter program and manual modification was made as needed; some cleanup could still be done on unneeded commons statements, as noted above. An additional problem arose in a subroutine REFINE which refines solutions to a single non-linear equation. This contains

convergence criteria in exceptional cases (which arise fairly often in SLACK) which depend on the details of real arithmetic. Two parameters (the maximum real number and a constant related to maximum precision) had to be modified since the VAX 8-byte real numbers are not the IEEE standard 8-byte reals used in virtually all current microcomputer compilers, whether for 8086 or 68000 series computers. The maximum real number for the IEEE format is readily available: about 1.797×10^{308} . The precision factor δ , defined as the smallest number for which $x \neq (1+\delta)x$ can be assured (i.e. is true for all x), is not so easily found. It may be estimated from the length of the fraction part of the number, but its precise value depends on the details of rounding which are not given in typical compiler documentation. Estimates put δ at 2^{-52} or 2.22×10^{-15} . A program was written to locate an "individual" δ by logarithmic bisection for randomly chosen numbers x in an interval; the maximum individual δ then forms the universal δ which will insure that the criterion is satisfied. Several runs for a variety of ranges lead to $\delta = 1.12 \times 10^{-15}$.

A working version has been produced. It's results need to be compared to those from the VAX and any discrepancies accounted for; some may be expected due to the difference in precision of the arithmetic (the IEEE standard has greater range than the VAX and about a digit less precision). The VAX version had been deleted due to disk space limitations during an extended period in which SLACK was not being used. It must either be restored from tape or recreated from source code, and a controlled set of comparisons run.

It must be noted that experience with the Microsoft compiler has been less than satisfactory. Fairly innocuous errors cause the computer to hang, requiring a coldboot. And although a working version has been created, the difference between that version and versions which fail to run (indeed hang the system) is unclear, seeming to reside in the options chosen on compiling (options chosen, indeed, to provide diagnostics on failure). Even when failing in a nondestructive way, the diagnostics provided are not very informative. Running with the Codeview debugger provided, though it is in many ways a very attractive tool, sometimes seems to generate failures or fail in different ways than without the debugger. Although all these problems arise occasionally with compilers on larger systems, the fragility of the Microsoft system supports the various word of mouth reports that it is not the preferred microcomputer compiler. (To be fair, most benchmarks show the Microsoft compiler producing moderately to substantially faster code than its rivals.) (Microsoft was initially chosen because of its attractive street price, about half that of other comparable compilers, a significant factor for personal acquisition.) If the problems experienced do not quickly resolve themselves, another compiler will be obtained; the two most generally favored are Lahey and Ryan McFarland, each with different advantages.

5.2.3 Modification Of SLACK Code

Only slight effort was devoted to modifications of the SLACK code to reflect the more general drag in the trajectory calculation. Most effort was spent in getting the existing code working on the microcomputer; modifications will be made in that setting. Several other factors require analysis and inclusion in the code: periodic updating of the drag forces; specification of the thruster positions and firing strategy; calculation of the tether area presented, and tether configuration for lift; computing the aerodynamic forces due to the RCS plumes.

6.0 PROBLEMS ENCOUNTERED DURING REPORTING PERIOD, G. GULLAHORN PI.

None

7.0 ACTIVITY PLANNED FOR NEXT REPORTING PERIOD, G. GULLAHORN PI.

7.1 Tether Applications Simulation Working Group Support

Preparing a final report on this task will be the first priority during the initial period of the next quarter. A major effort for the report will be preparing a uniform set of comparison plots for the various cases and simulators; this effort is complicated by the fact that the Imagen laser printer which has been used for plotting does not have sufficient memory to process some of the plots needed; a second printer exists with (presumably) adequate memory, but different scalings and fonts will require modification of the plotting schemes which had been developed. An alternative is the use of a microcomputer based graphics system, which would typically allow more convenient placement of labels, etc. A number of such systems are being evaluated to see if they will support the number of lines and data points needed for our purposes; a high resolution dot matrix printer is available. Work on this task, other than reporting, is essentially complete. Some few further efforts may be made as suggested in Quarterly 11.

7.2 Tether Aerodynamic Effect Of RCS Thruster Plume

The topic of aerodynamic forces on the tether due to the RCS plumes should be given further consideration, in particular to find an indication of the orthogonal (lift) forces. Experimental results may now exist. The relationship of the orthogonal force to the angle between the tether and the flow needs to be better understood.

Modification of the SLACK code will be continued. It is probable that a preliminary version, at least, can be completed in the next quarter.

The port of the SLACK code to the microcomputer will be studied and completed. The VAX version of SLACK will be resurrected, and comparison runs made. If the current compiler cannot be made to perform in a consistent and robust fashion, a new compiler may be acquired.

8.0 TECHNICAL ACTIVITY DURING REPORTING PERIOD AND PROGRAM STATUS, R.D. ESTES, PI

Most of our effort in the study of the plasma physics of hollow cathodes for use with tethered satellite systems in this reporting period was devoted to software development. This was carried out on one of the Macintosh II computer systems recently purchased by the group, partially out of funds from this contract. The development system used was Lightspeed C, an integrated system consisting of an editor, C compiler, and linker that can generate "stand-alone" Macintosh applications which can be run as regular Macintosh programs on any Macintosh computer. The system has full support for all of the features of the Macintosh computer that make it "user friendly", including menus, event-driven programs, and windows.

The program implemented was the one-dimensional electrostatic plasma simulation program ES1, originally developed by Birdsall and Langdon [1985] for use on a Cray-1 supercomputer. While this program does not include all aspects of the physics that may be important for the problem we are considering, it can serve as a useful starting point, especially for analyzing experiments conducted in plasma chambers, which are less complicated than the environment of a tethered satellite orbiting through the ionospheric magnetoplasma. A one-dimensional model is clearly insufficient to deal with the combined effects of orbital motion and the geomagnetic field, but we should be able to use the existing

subroutines to develop a two-dimensional computer code.

The original source code for ES1 is written in rather obscure Fortran for a Cray computer. It has been translated into transparent (hopefully--at least to a C programmer), modular C code. The use of the C language was dictated both by the programmer's preference and the greater ease with which the Macintosh system's "toolbox" can be accessed with existing C implementations. The new version should be much easier to modify than the original Fortran version as well.

This translation of the simulation program to a different language and a different system required a substantial effort, which we feel was well justified, since it gives us a version that we can not only use on our own computers but can share with other investigators using Macintosh systems. The programming effort was not confined to writing equivalent C code for the simulation program. It was also necessary to integrate this into the Macintosh environment. This required writing a number of i/o utility routines and graphics routines that make use of windows. For example, the "snapshot" plots that are periodically made and stored on microfiche in the original Cray version are stored as PICT files on the Macintosh hard disk. Software has been written to read these files and display them graphically in windows on the computer screen and to print them, as desired.

We are presenting the source code for this project, as it stands. It probably still contains a few bugs, but it has been tested for a number of simple cases, including the case of counterstreaming electrons. The file-handling and plotting

routines are working. The simulation code has so far only been run for cases with periodic boundary conditions, which are not applicable to a hollow cathode model. The graphical display code includes a novel feature. By selecting a menu item, the user can make the display windows "transparent", so that two graphs in two different windows (the one on top obscuring the one below it in the usual display) can be compared. Although we have not yet implemented it as a regular feature, the use of color to distinguish different particle populations has been demonstrated to be an easily programmed, effective analytical tool.

To facilitate running different cases of the simulation program, while at the same reducing the chance of introducing errors in the input data, we have developed a "front end" for the program that utilizes HyperCard, the object-based programming environment of the Macintosh. HyperCard is easier to demonstrate than to describe, but the figures that follow will hopefully clarify the basic ideas. Figure 7 shows (with annotations) the Macintosh screen display of the first "card" of our plasma simulation HyperCard "stack". All of the input parameters necessary to define the run, exclusive of the parameters characterizing the different particle species, are included on this card. That is, the number of species, number of time steps, etc. are displayed for the text file whose title is displayed in the box at the top of the card. Certain areas of the screen (card) are defined to be buttons ("Open", "Save", "Run", "nearest grid point", etc.). When the mouse button is clicked while the mouse-controlled cursor is within the defining area of one of these buttons, the action appropriate to that button is carried out. These

actions are programmable using the HyperTalk scripting language of HyperCard. For example, clicking on the "Run" button copies all of the input parameter values (as they are shown on the cards) to a selected text file (which can then be read by the simulation program) and launches the simulation program. The values of the input parameters are stored in "fields" that can be modified using the usual Macintosh text editing method, which relies upon selecting the text to be changed by means of the mouse. Only the contents of the fields can be modified in this way; the descriptive text is "locked".

The advantage of such a front-end is that the meaning of the input parameters can be spelled out in English, so that it is not necessary to remember what the equivalent variable name is in the computer program. All of the parameters can be displayed at once, allowing the complete picture of the computer run they represent to be taken in at a glance. This represents a great improvement over the usual alternatives, which consist of either changing the parameters within the source code and recompiling, making changes directly to the text input file (an error-prone method), or going through a tedious question and answer session with the computer via the screen and keyboard on every run. Buttons and fields are trivial to create within HyperCard, so modifications to the stack can be made as the program evolves with minimum programming effort. Figure 8 shows one of the cards that defines input parameters appropriate to a particular group of simulation particles.

ORIGINAL PAGE IS
OF POOR QUALITY

All input parameters can be changed by Macintosh editing

Mouse clicks on control buttons copy input data between HyperCard and the text file read by the program. Run button stores data and launches the simulation program.

Plasma simulation input data is in
DataFrame™ 20:LSC System:Plasma simulations:
simulation data

Identifying information: Fri, Jun 24, 1988 8:38 AM

number of species:	2	quantity	plot interval
number of time steps:	300	charge density (rho)	50
time step size (dt):	0.2	smoothed rho	50
length L (units of 2π):	1.	potential (phi)	50
number of grid points:	32	electric field	50
epsi (normally one):	1.0	phase space (x, vx)	50
background density:	0.0	velocity space(vx,vy)	0
		velocity distribution	50

kind of weighting:

☐ nearest grid point

☒ momentum conserving

☐ energy conserving

smoothing: a1 = 0.0 a2 = 0.0


external E: e0 = 0.0 Look at 4

omega0 = 0.0 modes.

Computing options selected with "radio buttons"

Names of variables and function of program flags can be spelled out in English to avoid confusion.

Figure 7



1 Plasma simulation input data is in
DataFrame™ 20:LSC System:Plasma simulations:
simulation data

Identifying information: Fri, Jun 24, 1988 8:38 AM

number of particles:	256
plasma frequency:	1.0
cyclotron freq.(sign):	0.0
charge/mass (sign):	-1.0
number of sub-groups:	1
drift velocity:	1.0
thermal velocity:	0.0

MODE EXCITATION	
mode number:	1
size of x perturbation:	.001
size of v perturbation:	0.0
phase in x (thetax):	0.0
phase in v (thetav):	0.0

type of velocity generation:
☐ random number
☒ inverse distribution
 exponent for v dist.(nv2): 0

first group:
v0 = 1.

Navigation icons: left arrow, right arrow, up arrow, down arrow

Figure 8

Details of the simulation program can be found in the Birdsall and Langdon reference. We now present the source code for our version, module by module.

References

1. Birdsall, Charles K. and A. Bruce Langdon. Plasma Physics via Computer Simulation, McGraw-Hill Book Company (1985).

I. electrostat.c

This module includes the main control routine for the simulations. In “main()” the initialization routines are called and the time integration loop is gone through. The module also contains the event loop, which responds to mouse events (menu selections, etc.) whenever the program goes into the interactive mode. The routines to make “snapshot” and time-history graphs are included.


```

#include "plasma.h"

int my_mask = mDownMask + activMask;
Boolean valid_event;
FILE *output_file;
FILE *data_read;

double TWOPI;
double L, dx, dt;
double elapsed_time;
double ael, epsi, rho0, a1, a2, e0, w0;

double *x, *vx, *vy;
double rho[NGIM], phi[NGIM], e[NGIM];
double ese[NTH1], nms[NSPM];
/*
    double esem[MMAX][NTH1];
    double kes[NSPM][NTH1], pxs[NSPM][NTH2];
*/
double *esem[MMAX];
double *kes[NSPM], *pxs[NSPM];

double scratch[NGIM], fft1[NGTWO];
double x_points[NGMAX];
double *L_points;
double f_of_v[25];
max_min xv_limits[2];
double vte;

int mplot[MMAX];
int ng, iw;
int nsp, ntp;
int it, ith1, irho, irhos, iphi, ie, ixvx, ivxvy, ifvx;

extern WindowPtr draw_window[4];
extern PicHandle draw_picture[4];
extern      int rho_file, rhos_file, e_file, phi_file, vxvy_file, vxv_file,
              fv_file, history_file;
extern long header[128];
extern int my_ref;

extern void do_preliminaries();
extern void initialize();
extern void get_fields(), set_v(), bin_fv(), trace_plot(), draw_plot();
extern void accelerate(), advance(), do_mouse_down(), do_update();
extern void scatter_plot(), do_activate(), do_xv_plot(), do_ee_update();
extern void under_update();

void event_loop(), record(), finish(), show(), show1();
void save_fv(), save_graph();
double ranf();

Boolean RETURN;
extern Boolean GROW, DRAG, UNDER_ONLY, INI, INP;
extern Rect overlap_rect;

```

```

/*-----*/
main()
{
    double p_total = 0.;
    double ke_total = 0.;
    double ms[NSPM], qs[NSPM], ts[NSPM];
    double te = 0.;
    double vl, vu, vmu;
    int ins[NSPM];
    register int ith = 0;
    register int is;
    int nt;
    int nmodes;
    max_min fv_limits[2], all_xv_lims[2], vxvy_limits[2];
    do_preliminaries(ins, &nt, &nmodes);
    for( is = 0; is < nsp; is++)
    {
        initialize(ins[is], &ins[is + 1], &ms[is], &qs[is], &ts[is],
                    &nms[is]);
    }
    get_fields(0);
    for( is = 0; is < nsp; is++)
    {
        set_v(ins[is], ins[is + 1], qs[is], ms[is], ts[is], pxs[is]);
    }
    SysBeep(10);
    event_loop();
    fprintf(output_file, " time      ese      p_total      ke1");
    fprintf(output_file, "      ke2      te");
    while(it < nt)
    {
        if(ixvx != 0 && (it % ixvx == 0))
        {
            save_graph(xvx_file, x, vx, all_xv_lims, ins[nsp],
                        draw_window[1], &draw_picture[1],
                        "\pPhase Space of All", TRUE, do_xv_plot);
        }
        if(ifvx != 0 && (it % ifvx == 0))
        {
            save_fv(fv_limits, ins[1], 13);
        }
        if(ts[0] != 0. && ivxvy != 0 && (it % ivxvy == 0))
        {
            save_graph(vxvy_file, vx, vy, vxvy_limits, ins[1],
                        draw_window[0], &draw_picture[0], "\pVx-Vy space",
                        TRUE, scatter_plot);
        }
        p_total = 0.;
        ke_total = 0.;
        for( is = 0; is < nsp; is++)
        {
            accelerate(ins[is], ins[is + 1], qs[is], ms[is], ts[is],
                        &pxs[is][ith + 1], &kes[is][ith]);
            p_total += pxs[is][ith + 1];
            ke_total += kes[is][ith];
        }
        for( is = 0; is < nsp; is++)
        {
            advance(ins[is], ins[is + 1], qs[is]);
        }
        te = ke_total + ese[ith];
        fprintf(output_file, "\n%10.3e %10.3e %10.3e %10.3e %10.3e %10.3e",
                elapsed_time, ese[ith], p_total, kes[0][ith], kes[1][ith],
                te);
        /*if(ith == NTH)

```

```

        {    record();
        }*/
        t_points[it] = it * dt;
        it++;
        elapsed_time += dt;
        ith = it - ithl;
        get_fields(ith);
    }
    /*event_loop();*/
    /*scatter_plot(x, vx, xv_limits, ins[1], draw_window[2],
                   &draw_picture[2], "\pVx versus X phase space", TRUE);*/
    record(nt, nmodes);
    event_loop();
    finish();
}
/*-----*/
void
event_loop()
{
    EventRecord my_event;
    WindowPtr event_window;
    Boolean valid;
    RETURN = FALSE;
    while (!RETURN)
    {
        SystemTask();
        valid = GetNextEvent(everyEvent, &my_event);
        if (!valid)
        {
            continue;
        }
        switch(my_event.what)
        {
            case nullEvent:
                break;
            case mouseDown:
                if(UNDER_ONLY != GROW)
                {
                    break;
                }
                do_mouse_down(&my_event);
                break;
            case mouseUp:
                break;
            case keyDown:
                finish();
            case keyUp:
            case autoKey:
                break;
            case updateEvt:
                event_window = (WindowPtr)my_event.message;
                if(event_window == FrontWindow() &&
                   GetWRefCon(event_window) < 0L && INF)
                {
                    if(UNDER_ONLY)
                    {
                        under_update(&my_event);
                    }
                    else
                    {
                        do_see_update(&my_event);
                    }
                }
            }
        }
    }
}

```

```

        else
        {
            do_update(&my_event);
            UNDER_ONLY = DRAG = GROW = FALSE;
        }
        break;
    case diskEvt:
        break;
    case activateEvt:
        do_activate(&my_event);
        break;
    case networkEvt:
    case driverEvt:
    case applEvt:
    case app2Evt:
    case app3Evt:
    case app4Evt:
    default:
        break;
    }
}

/*-----*/
void
finish()
{
    if(my_ref != -999)
    {
        PSClose(my_ref);
    }
    ExitToShell();
}

/*-----*/
void
save_fv(fv_limits, n, nbins)
register max_min *fv_limits;
register int n, nbins;
{
    static int call_number = 0;
    register int save_error;
    bin_fv(vx, f_of_v, 0., 2., n, nbins);
    if(call_number == 0)
    {
        limit_calc(x_points, f_of_v, fv_limits, nbins);
    }
    trace_plot(x_points, f_of_v, fv_limits, nbins, draw_window[0],
               &draw_picture[0], "pVelocity Distribution", TRUE);
    draw_plot(draw_window[0], draw_picture[0], "pVelocity Distribution");
    if(call_number == 0)
    {
        call_number++;
        if((save_error = write(fv_file, (char *)&header, (unsigned)512))
           != 512)
        {
            SysBeep(20);
        }
    }
    if((save_error = write(fv_file, (char *)&(draw_picture[0]),
                          (unsigned)((*(draw_picture[0]))->picSize))) == -1)
    {
        SysBeep(10);
    }
}

```

```

/*-----*/
void
save_graph(iplot, x, y, xy_limits, n, graph_window, graph_pic,
           graph_label, time_test, grapher)
    register int iplot, n;
    register double *x, *y;
    register max_min *xy_limits;
    WindowPtr graph_window;
    PicHandle *graph_pic;
    char *graph_label;
    Boolean time_test;
    void (*grapher)();
{
    static int counter[14];
    int save_error;
    /*if(counter[iplot] == 0 || iplot == history_file) HERE'S THE SCALING*/
    {
        limit_calc(x, y, xy_limits, n);
    }
    (*grapher)(x, y, xy_limits, n, graph_window, graph_pic, graph_label,
              time_test);
    /*draw_plot(graph_window, *graph_pic, graph_label);*/
    if(counter[iplot] == 0)
    {
        counter[iplot]++;
        if((save_error = write(iplot, (char *)&header, (unsigned)512))
            != 512)
        {
            SysBeep(20);
        }
    }
    if((save_error = write(iplot, (char *)&graph_pic,
                          (unsigned)((**graph_pic)->picSize))) == -1)
    {
        SysBeep(10);
    }
}
/*-----*/

void
record(nt, nmodes)
    register int nt, nmodes;
{
    register char *mode_label = NewPtr(30);
    register int i;
    max_min mode_limits[2];
    for( i = 0; i < nmodes; i++)
    {
        sprintf(mode_label, "energy in mode %d", i);
        CtoPtr(mode_label);
        save_graph(history_file, t_points, esem[i], mode_limits, nt,
                  draw_window[i % 4], &draw_picture[i % 4], mode_label,
                  TRUE, trace_plot);
    }
    save_graph(history_file, t_points, esem, mode_limits, nt,
              draw_window[2], &draw_picture[2], "\pelectrostatic energy",
              TRUE, trace_plot);
    save_graph(history_file, t_points, kes[0], mode_limits, nt,
              draw_window[3], &draw_picture[3], "\pkinetic energy",
              TRUE, trace_plot);
}
/*-----*/

double
ranf()

```

```

(    double rand_value;
    double randi;
    int first;
    randi = 1. * Random();
    rand_value = (randi + 32767.) / 65534.;
    return (rand_value);
)
/*-----*/
void
show(x, y, n)
    register double *x, *y;
    register int n;
    register int i;
(    for(i = 0; i < n; i++)
    (    fprintf(output_file, "\n%e %e", x[i], y[i]);
    )
)
/*-----*/
void
show1(x, n)
    register double *x;
    register int n;
    register int i;
(    for(i = 0; i < n; i++)
    (    fprintf(output_file, "\n%d %e", i, x[i]);
    )
)

```

II. `plasmastart.c`

This module includes routines to set up Macintosh windows and menus that are then available for use throughout the program. It also contains the routines for reading the input data file and initializing the relevant program variables. The first part of the actual simulation code, which creates the simulation particles according to the data in the input file, is also found in this module.

```

#include "plasma.h"

WindowRecord draw_record[4];
WindowPtr draw_window[4];
ControlHandle bars[4][2];
ControlHandle lookup_control();

Rect drag_rect, grow_bounds;
PicHandle draw_picture[4];
Rect big_rect, biggest_rect = ( 0, 0, 756, 576);
Rect clipper;
int rho_file, rhos_file, e_file, phi_file, vxvy_file, vx_file;
int fv_file, history_file;

extern double TWOPI;
extern double L, dx, dt;
extern double elapsed_time;
extern double ael, epsi, rho0, a1, a2, e0, w0;
extern OSType my_creator, my_type;

extern double *x, *vx, *vy;
extern double rho[NGIM], phi[NGIM], e[NGIM];
extern double ese[NTH1], nms[NSPM];
/*
    double esem[MMAX][NTH1];
    double kes[NSPM][NTH1], pxs[NSPM][NTH2];
*/
extern double *esem[MMAX];
extern double *kes[NSPM], *pxs[NSPM];

extern double scratch[NGIM], ffti[NGTWO];
extern double x_points[NGMAX], *t_points;
extern double f_of_v[25];
extern max_min xv_limits[2];
extern double vte;

extern int mplot[MMAX];
extern int ng, iw;
extern int nsp, ntp;
extern int it, ith1, irho, irhos, iphi, ie, ixvx, ivxvy, ifvx;

extern double ranf();
extern void accelerate(), move_bars(), fill_sine_table();

extern FILE *output_file;
extern FILE *data_read;
extern Boolean RETURN;
extern void show();

void fill_menus(), define_arrays(), set_defaults(), read_in_values();
void make_files(), zero_all(), initialize(), do_even_load();
void do_ordered_load(), scramble(), do_rotation(), copy_group();
void add_maxwellian(), add_perturbation(), set_rho(), field_init(), set_v();
void do_inits(), make_windows(), set_parameters(), make_window();
void init_bar(), switch_control(), do_preliminaries(), fill_x();

```



```

/*-----*/
void
do_preliminaries(ins, nt, nmodes)
int *ins, *nt, *nmodes;
{
    int *dummy = (int*)(&thePort);
    do_inits();
    define_arrays();
    dummy[-63] = 331;
    TWOPI = 2. * PI;
    ntp = 100;
    it = ithl = 0;
    elapsed_time = 0.;
    ins[0] = 0;
    zero_all();
    read_in_values(nt, nmodes);
    make_files();
    make_windows();
    L *= TWOPI;
    dx = L / ng;
    fill_x(ng + 1);
    fill_sine_table();
}
/*-----*/

void
do_inits()
{
    InitGraf(&thePort);
    InitFonts();
    InitWindows();
    TEInit();
    InitDialogs(OL);
    MaxApplZone();
    InitMenus();
    FlushEvents( everyEvent, 0 );
    InitCursor();
    MoreMasters();
    MoreMasters();
    MoreMasters();
    MoreMasters();
    MoreMasters();
    MoreMasters();
    Stdio_MacInit(TRUE);
    set_parameters();
    fill_menus();
}
/*-----*/

void
make_windows()
{
    register int i;
    /*big_rect.left = 0;
    big_rect.top = 0;
    big_rect.right = screenBits.bounds.right;
    big_rect.bottom = (screenBits.bounds.right * 10);
    big_rect.bottom /= 8;*/
    big_rect = biggest_rect;
    clipper = screenBits.bounds;
    clipper.bottom = biggest_rect.bottom;
}

```

```

/*big_rect = screenBits.bounds;*/
for(i = 0; i < 4; i++)
{
    draw_window[i] = (WindowPtr)&draw_record[i];
    make_window(draw_window[i], GRAPH_WINDOW);
}
/*-----*/
void
set_parameters()
{
    drag_rect = thePort->portRect;
    SetRect(&grow_bounds, 64, 64, thePort->portRect.right,
        thePort->portRect.bottom);
}
/*-----*/
void
make_window(new_window, window_id)
register WindowPtr new_window;
int window_id;
{
    ControlHandle my_scroll;
    new_window = GetNewWindow(window_id, new_window, -1L);
    SetPort(new_window);
    my_scroll = GetNewControl(V_SCROLL, new_window);
    SetCRefCon(my_scroll, (long)V_SCROLL);
    my_scroll = GetNewControl(H_SCROLL, new_window);
    SetCRefCon(my_scroll, (long)H_SCROLL);
    move_bars(new_window);
    init_bar(new_window, (long)H_SCROLL, 0, 50);
    init_bar(new_window, (long)V_SCROLL, 0, 50);
    DrawGrowIcon(new_window);
}
/*-----*/
void
init_bar(window, id, value, range)
register WindowPtr window;
register int value, range;
long id;
{
    ControlHandle control = lookup_control(window, id);
    if((range) HiliteControl(control, 255);
    else
    {
        SetCtlMin(control, 0);
        SetCtlMax(control, range);
        SetCtlValue(control, value);
        HiliteControl(control, 0);
        InvalRect(&(*control)->controlRect);
    }
}
/*-----*/
void
switch_control(window, id, hilite)
WindowPtr window;
long id;
int hilite;
{
    ControlHandle control = lookup_control(window, id);
    HiliteControl(control, hilite);
}

```

```

/*-----*/
ControlHandle
lookup_control(window, id)
WindowPtr window;
long id;
(
    long label;
    ControlHandle control = ((WindowPeek)window)->controlList;
    while (control)
    (
        label = GetCRefCon(control);
        if (label == id) break;
        control = (*control)->nextControl;
    )
    return control;
)
/*-----*/

void
fill_menus()
(
    char *apple_name = NewPtr(20);
    MenuHandle menu;
    *apple_name = (char)1;
    *(apple_name + 1) = (char)appleMark;
    menu = NewMenu(APPLE_MENU, apple_name);
    AppendMenu(menu, "\pAbout Plasma ...");
    AppendMenu(menu, "\p(-");
    AddResMenu(menu, DRVr);
    InsertMenu(menu, 0);
    InsertMenu(GetMenu(FILE_MENU), 0);
    InsertMenu(GetMenu(EDIT_MENU), 0);
    InsertMenu(menu = GetMenu(PLASMA_MENU), 0);
    CtoPstr(apple_name = "I Shrink to Fit");
    apple_name[2] = (char)checkMark;
    AppendMenu(menu, apple_name);
    DrawMenuBar();
    DisableItem(GetMenu(EDIT_MENU), 0);
)
/*-----*/

void
define_arrays()
(
    register int i;
    x = (double *)NewPtr(sizeof(double[NPAR]));
    vx = (double *)NewPtr(sizeof(double[NPAR]));
    vy = (double *)NewPtr(sizeof(double[NPAR]));
    for( i = 0; i < NSPM; i++)
    (
        pxs[i] = (double *)NewPtr(NTH2 * sizeof(double));
        kes[i] = (double *)NewPtr(sizeof(double[NTH1]));
    )
    t_points = (double *)NewPtr(sizeof(double[NTH1]));
    for( i = 0; i < MMAX; i++)
    (
        esem[i] = (double *)NewPtr(NTH1 * sizeof(double));
    )
)
/*-----*/

void
set_defaults(nt)
int *nt;
(
    register int i;

```

```

    nsp = 1;
    L = TWOPI;
    dt = .2;
    *nt = 10;
    epsi = 1.;
    ng = 32;
    iw = 2;
    a1 = a2 = e0 = w0 = 0.;
    irho = irhos = iphi = ie = ixvx = ivxvy = ifvx = 20;
    for(i = 0; i < MMAX; i++)
    {
        mplot[i] = 0;
    }
}

/*-----*/
void
read_in_values(nt, nmodes)
register int *nt, *nmodes;
{
    register int i;
    char ch_read =(char)0;
    data_read = fopen("simulation data", "r");
    while(ch_read != ("\n"))
    {
        fscanf(data_read, "%c", &ch_read);
    }
    fscanf(data_read, "%d\n%d\n%lf\n%lf\n%lf\n", &nsp, nt, &dt, &L, &ng);
    fscanf(data_read, "%lf\n%lf\n%lf\n%lf\n%lf\n%lf\n%lf\n%lf\n",
        &epsi, &rho0, &iw, &irho, &irhos, &iphi, &ie, &ixvx, &ivxvy,
        &ifvx);
    fscanf(data_read, "%lf\n%lf\n%lf\n%lf\n%lf\n", &a1, &a2, &e0, &w0,
        nmodes);
    for(i = 0; i < *nmodes; i++)
    {
        mplot[i] = 1;
    }
}

/*-----*/
void
make_files()
{
    register int open_mode = O_RDWR + O_CREAT + O_BINARY;
    output_file = fopen("run_file", "w+");
    my_type = 'PLOT';
    my_creator = 'PLAS';
    if(irhos != 0)
    {
        rhos_file = open("smoothed rho.plot", open_mode);
    }
    if(irho != 0)
    {
        rho_file = open("rho.plot", open_mode);
    }
    if(ie != 0)
    {
        e_file = open("electric field.plot", open_mode);
    }
    if(iphi != 0)
    {
        phi_file = open("phi.plot", open_mode);
    }
    if(ivxvy != 0)
    {
        vxvy_file = open("vxvy.plot", open_mode);
    }
    if(ixvx != 0)

```



```

    *m = *q / qm;
    *nm = n * (*m);
    ngr = n / nlg;
    Lg = L / nlg;
    ddx = L / n;
    do_even_load(ill, ngr, v0, ddx);
    if(ORDERED)
    {
        do_ordered_load(ill, *il2, n, nv2, ngr, vt2, &skip);
        if(!skip)
        {
            scramble(Lg, ddx, ill, ngr);
        }
    }
    if(MAGNETIZED && !skip)
    {
        do_rotation(ngr, ill, Lg);
    }
    if(ANOTHER)
    {
        copy_group(ill, ngr, n, Lg, wc);
    }
    if(RANDOM)
    {
        add_maxwellian(n, ill, vt1, wc);
    }
    add_perturbation(n, ill, mode, x1, v1, thetax, thetav);
    set_rho(ill, *il2, *q, n * (*q) / L);
    /*showl(rho, ng + 1);*/
}
/*-----*/
void
do_even_load(ill, ngr, v0, ddx)
register int ill, ngr;
double v0, ddx;
{
    register int i, il;
    double x0;
    for( i = 0; i < ngr; i++)
    {
        il = i + ill;
        x0 = (i * 1. + .5) * ddx;
        x[il] = x0;
        vx[il] = v0;
    }
}
/*-----*/
void
do_ordered_load(ill, il2, n, nv2, ngr, vt2, skip)
int ill, il2, nv2;
register int n, ngr;
double vt2;
Boolean *skip;
{
    register int i, il, j;
    double vmax, dv, vv, vvnv2, fv, df;
    vmax = .5 * vt2;
    dv = 2. * vmax / (n - 1);
    vvnv2 = 1.;
    x[ill] = 0.;
    for(i = 1; i < n; i++)
    {
        vv = ((i - .5) * dv - vmax) / vt2;
        if(nv2 != 0)
        {
            vvnv2 = pow(vv, nv2);
        }
    }
}

```

```

    )
    fv = vv*vv2 * exp( -.5 * vv * vv);
    il = i + ill;
    x[il] = x[il - 1] + my_max(fv, .0);
}
df = x[il] / ngr;
il = ill;
j = ill;
for( i = 0; i < ngr; i++, il++)
(
    fv = (i + .5) * df;
    while(fv >= x[j + 1])
    {
        j++;
        if(j > il2 - 2)
        {
            *skip = TRUE;
            return;
        }
    }
    vv = dv * (j - ill + (fv - x[j]) / (x[j + 1] - x[j])) - vmax;
    vx[il] += vv;
)
}
/*-----*/

void
scramble(Lg, ddx, ill, ngr)
double Lg, ddx;
register int ill, ngr;
(
    register int i, il;
    double xs = 0.;
    double xsl;
    for( i = 0; i < ngr; i++)
    {
        il = i + ill;
        x[il] = xs * Lg + .5 * ddx;
        /* write(); */
        xsl = 1.;
        while(xs >= 0.)
        {
            xsl *= .5;
            xs -= xsl;
        }
        xs += 2. * xsl;
    }
)
/*-----*/

void
do_rotation(ngr, ill, Lg)
register int ngr, ill;
double Lg;
(
    register int i, il;
    double vv, theta;
    for( i = 0; i < ngr; i++)
    {
        il = ill + i;
        vv = vx[il];
        theta = TWOPI * x[il] / Lg;
        vx[il] = vv * cos(theta);
        vy[il] = vv * sin(theta);
    }
)

```

```

/*-----*/
void
copy_group(ill, ngr, n, Lg, wc)
int ill, ngr;
register int n;
double Lg, wc;
{
    register int i, j, il, i2;
    double xs = 0.;
    for (i = ngr; i < n; i += ngr)
    {
        xs += Lg;
        for (j = 0; j < ngr; j++)
        {
            il = j + ill;
            i2 = il + i;
            x[i2] = x[i1] + xs;
            vx[i2] = vx[i1];
            if(MAGNETIZED)
            {
                vy[i2] = vy[i1];
            }
        }
    }
}

/*-----*/
void
add_maxwellian(n, ill, vt1, wc)
register int n, ill;
double vt1, wc;
{
    register int i, il, j;
    for (i = 0; i < n; i++)
    {
        il = ill + i;
        for (j = 0; j < 12; j++)
        {
            if(MAGNETIZED)
            {
                vy[i1] += vt1 * (ranf() - .5);
            }
            vx[i1] += vt1 * (ranf() - .5);
        }
    }
}

/*-----*/
void
add_perturbation(n, ill, mode, x1, v1, thetax, thetav)
register int n, ill, mode;
double x1, v1, thetax, thetav;
{
    register int i, il;
    double theta;
    for (i = 0; i < n; i++)
    {
        il = ill + i;
        theta = TWOPI * mode * x[i1] / L;
        x[i1] += x1 * cos(theta + thetax);
        vx[i1] += v1 * sin(theta + thetav);
    }
}

/*-----*/
void
set_rho(il, iu, q, rhos)
register int il, iu;
double q, rhos;

```



```

(   register int i, j;
    double dxi, xn, drho, qdx;
    qdx = q / dx;
    dxi = 1. / dx;
    xn = ng;
    if(FIRST_GROUP)
    {   for( i = 0; i < ng; i++)
        {   rho[i] = rho0;
            }
        rho[ng] = 0.;
    }
/*   show(x_points, rho, ng + 1);*/
    rho0 -= rhos;
    for( i = 0; i < ng; i++)
    {   rho[i] -= rhos;
    }
/*   show(x_points, rho, ng + 1);*/
    switch(iw)
    {   case ZERO_ORDER:
        for( i = 1; i < iu; i++)
        {   x[i] *= dxi;
            if(x[i] < 0.)
            {   x[i] += xn;
            }
            if(x[i] >= xn)
            {   x[i] -= xn;
            }
            j = x[i] + .5;
            rho[j] += qdx;
        }
        break;
        case MOMENTUM:
        case ENERGY:
            for( i = 1; i < iu; i++)
            {   x[i] *= dxi;
                if(x[i] < 0.)
                {   x[i] += xn;
                }
                if(x[i] >= xn)
                {   x[i] -= xn;
                }
                j = x[i];
                drho = qdx * ( x[i] - j);
                rho[j] += (qdx - drho);
                rho[j + 1] += drho;
            }
            break;
    }
/*   show(x_points, rho, ng + 1);*/
}
/*-----*/
void
field_init(sm, ksqi, ng2)
register double *sm, *ksqi;
register int ng2;

```

```

(   register int i;
    double kdx2;
    for( i = 0; i < ng2; i++)
    {   kdx2 = (PI / ng) * (i + 1);
        sm[i] = exp(a1 * pow(sin(kdx2), 2.) - a2 * pow(tan(kdx2), 4.));
        ksq[i] = (pow(sm[i] * dx / (2. * sin(kdx2)), 2.) / epsi;
    }
)

/*-----*/
void
set_v(il, iu, q, m, t, p)
double q, m, t, *p;
register int il, iu;
{   double dtdx = dt / dx;
    double c, s, vxx;
    register int i;
    if(NEED_ROTATION)
    {   c = 1. / sqrt(1. + t * t);
        s = c * t;
        for ( i = il; i < iu; i++)
        {   vxx = vx[i];
            vx[i] = c * vxx + s * vy[i];
            vy[i] = -s * vxx + c * vy[i];
            vy[i] *= dtdx;
        }
    }
    for ( i = il; i < iu; i++)
    {   vx[i] *= dtdx;
    }
    accelerate(il, iu, -.5 * q, m, 0., p, scratch);
}

```

III. `plasmafiles.c`

This module consists of the routines to create, read, display, and print graphics files: snapshots made at selected intervals or time histories made the end of the simulations.

```

#include "plasma.h"
int dirty;
Size file_size;
long header[128];
Point sfg_where = (90, 82);
SFReply reply;
Str255 picture_name;
int my_ref = -999;
PicHandle file_picture;

extern FILE *output_file;
extern Boolean MORE_PLOTS;
extern Rect big_rect, clipper;
extern void show_scrolls(), hide_scrolls(), erase_grow(), do_window_care();

void do_new(), do_save_as(), do_open(), do_save(), show_error();
void do_read(), do_print_close(), handle_error();
/*-----*/
do_print()
{
    GrafPtr save_graf, save_print;
    WindowPtr the_window = FrontWindow();
    TPrPort print_port;
    THPrint my_print;
    TPrStatus p_status;
    TSetRslBlk image_set;
    Boolean valid;
    int *print_rect;
    int i;
    PicHandle draw_picture = (PicHandle)my_abs(GetWRefCon(FrontWindow()));
    GetPort(&save_graf);
    image_set.iOpCode = setRslOp;
    image_set.iXRsl = 144;
    image_set.iYRsl = 144;
    PrOpen();
    if(PrintErr != noErr)
    {
        return PrintErr;
    }
    my_print = (THPrint)NewHandle(sizeof(TPrint));
    PrintDefault(my_print);
    valid = PrValidate(my_print);
    image_set.hPrint = my_print;
    PrGeneral(&image_set);
    if(image_set.iError != 0)
    {
        SysBeep(10);
        SysBeep(10);
    }
    print_port = PrOpenDoc(my_print, 0L, 0L);
    if(PrintErr != noErr)
    {
        return PrintErr;
    }
    GetPort(&save_print);
    SetPort(the_window);
    hide_scrolls(the_window);
    erase_grow(the_window);
    SetPort(save_print);
    valid = PrJobDialog(my_print);
}

```

```

if(!valid)
{
    SetPort(the_window);
    SysBeep(10);
    show_scrolls(the_window);
    DrawGrowIcon(the_window);
    SetPort(save_graf);
    return;
}
valid = PrStdDialog(my_print);
if(!valid)
{
    SetPort(the_window);
    show_scrolls(the_window);
    DrawGrowIcon(the_window);
    SetPort(save_graf);
    return;
}
PrOpenPage(print_port, 0L /*&big_rect*/); /* this choice means no scaling */
if(PrintErr != noErr)
{
    do_print_close(print_port, save_graf);
    return PrintErr;
}
print_rect = (int *)&(print_port->gPort.portRect);
for( i = 0; i < 4; i++)
{
    fprintf(output_file, "\n%d", *(print_rect++));
}
DrawPicture(draw_picture, &(print_port->gPort.portRect)
/*&big_rect*/);
if(PrintErr != noErr)
{
    do_print_close(print_port, save_graf);
    return PrintErr;
}
PrClosePage(print_port);
PrCloseDoc(print_port);
if((*my_print)->prJob.bJDocLoop == bSpoolLoop)
{
    if(PrintErr != noErr)
    {
        PrClose();
        SetPort(save_graf);
        return PrintErr;
    }
    PrPicFile(my_print, 0L, 0L, 0L, &p_status);
}
PrClose();
SetPort(the_window);
show_scrolls(the_window);
DrawGrowIcon(the_window);
SetPort(save_graf);
}
/*-----*/
void
do_print_close(print_port, the_save)
    TPrPort print_port;
    GrafPtr the_save;
{
    PrClosePage(print_port);
    PrCloseDoc(print_port);
    PrClose();
    SetPort(the_save);
}

```

```

)
/*-----*/
void
handle_error(the_error)
int the_error;
{
}
/*-----*/
void
do_new()
{
    /*SetWTitle(draw_window[0], "\pUntitled");
    ShowWindow(draw_window[0]);
    dirty = 0;*/
}
/*-----*/
void
do_save_as()
{
    register WindowPtr plot_window = FrontWindow();
    register int i;
    Str255 the_volume;
    long file_count;
    register PicHandle plot_picture;
    long header_count = 512;
    if(plot_window == 0L) return;
    plot_picture = (PicHandle)GetWRefCon(plot_window);
    file_size = file_count = GetHandleSize(plot_picture);
    picture_name[0] = 0;
    hide_scrolls(plot_window);
    erase_grow(plot_window);
    SFPutFile(sfg_where, "\pSave File As ...", picture_name, 0L, &reply);
    show_scrolls(plot_window);
    DrawGrowIcon(plot_window);
    if(reply.good)
    {
        if((i = FSOpen(reply.fName, reply.vRefNum, &my_ref)) != fnfErr)
        {
            ;
        }
        else
        {
            i = Create(reply.fName, reply.vRefNum, '????', 'PICT');
            i = FSOpen(reply.fName, reply.vRefNum, &my_ref);
        }
        SetWTitle(plot_window, reply.fName);
        i = FSWrite(my_ref, &header_count, *header);
        HLock(plot_picture);
        i = FSWrite(my_ref, &file_count, *plot_picture);
        HUnlock(plot_picture);
        if(i != noErr)
        {
            show_error(i, 30);
        }
        else
        {
            if((i = GetVol((StringPtr)the_volume, &reply.vRefNum))
               != noErr)
            {
                show_error(i, 40);
            }
            if((i = FSClose(my_ref)) != noErr)
            {
                show_error(i, 50);
            }
        }
    }
}

```

```

        if((i = FlushVol((StringPtr)the_volume, reply.vRefNum))
           != noErr)
        {
            show_error(i, 60);
        }
    }
}
/*-----*/
void
show_error(type_error, which_call)
int type_error, which_call;
{
    SysBeep(which_call * 10);
    fprintf(output_file, "\nerror = %d on call number %d", type_error,
            which_call);
}
/*-----*/
void
do_open(new_window, new_picture)
WindowPtr new_window;
PicHandle *new_picture;
{
    SFTypelist my_types;
    WindowPtr first_window;
    register int i;
    long header_count = 512;
    MORE_PLOTS = FALSE;
    my_types[0] = 'PLOT';
    if(my_ref != 999)
    {
        i = FSClose(my_ref);
        my_ref = -999;
    }
    if((first_window = FrontWindow()) != 0L)
    {
        hide_scrolls(first_window);
        erase_grow(first_window);
    }
    SFGetFile(sfg_where, "\p", 0L, 1, my_types, 0L, &reply);
    if(first_window != 0L)
    {
        show_scrolls(first_window);
        DrawGrowIcon(first_window);
    }
    if(reply.good)
    {
        if((i = FSOpen(reply.fName, reply.vRefNum, &my_ref)) != noErr)
        {
            show_error(i, 1);
            return;
        }
        if((i = FSRead(my_ref, &header_count, &header)) != noErr)
        {
            show_error(i, 2);
            return;
        }
        do_read(new_window, new_picture, reply, my_ref);
    }
    else
    {
        do_window_care();
    }
}
/*-----*/
void

```

```

do_read(new_window, new_picture, the_reply, the_ref)
    register WindowPtr new_window;
    PicHandle *new_picture;
    SFReply the_reply;
    register int the_ref;
    (
        register int i;
        int j;
        long pic_count = 2L;
        WindowPtr first_window;
        MORE_PLOTS = TRUE;
        if((i = FSRead(the_ref, &pic_count, &j)) != noErr)
            (
                if(i = eofErr)
                    (
                        if((first_window = FrontWindow()) != 0L)
                            (
                                hide_scrolls(first_window);
                                erase_grow(first_window);
                            )
                        NoteAlert(ENDALERT, 0L);
                        if(first_window != 0L)
                            (
                                show_scrolls(first_window);
                                DrawGrowIcon(first_window);
                            )
                        i = FSClose(the_ref);
                        my_ref = -999;
                        MORE_PLOTS = FALSE;
                    )
                return;
            )
        *new_picture = (PicHandle)NewHandle((long)j);
        HLock(*new_picture);
        if((i = SetFPos(the_ref, fsFromMark, -2L)) != noErr)
            (
                show_error(i, 4);
                return;
            )
        pic_count = (long)j;
        if((i = FSRead(the_ref, &pic_count, **new_picture)) != noErr)
            (
                if(i = eofErr)
                    (
                        NoteAlert(ENDALERT, 0L);
                        i = FSClose(the_ref);
                        my_ref = -999;
                        MORE_PLOTS = FALSE;
                    )
            )
        )
        SetWRefCon(new_window, (long)*new_picture);
        SetPort(new_window);
        ClipRect(&clipper);
        EraseRect(&clipper);
        SetWTitle(new_window, &the_reply.fName);
        SelectWindow(new_window);
        do_window_care();
        ShowWindow(new_window);
        DrawPicture(*new_picture, &big_rect/*&new_window->portRect*/ );
        HUnlock(*new_picture);
        /*DrawControls(new_window);
        DrawGrowIcon(new_window);*/
        /* THE FOLLOWING CODE IS BASICALLY RIGHT FOR DEFINING A BITMAP
        PICTURE OF THE GRAPH, BUT IT ISN'T INTEGRATED WITH THE REST

```


OF THE PROGRAM. LOTS OF BIZARRE DRAWING RESULTS IN TRANSPARENT
MODE.

```
*new_picture = OpenPicture(&big_rect);
CopyBits(&new_window->portBits, &new_window->portBits, &big_rect,
        &big_rect, srcOr, 0L);
ClosePicture();
SetWRefCon(new_window, (long)*new_picture);
*/
}
    void
do_save()
{
}
```

IV. plotting.c

This module contains routines to make scatter and line plots, which are stored as "pictures" available for display through the routines found in `plasmafiles.c`.

```

#include "plasma.h"

extern Rect big_rect, clipper;
extern FILE *output_file;
extern double elapsed_time;

extern double L, dx, dt;

void scatter_plot(), limit_calc(), trace_plot(), line_it(), move_it();
void map_point(), make_point(), bin_fv(), draw_plot(), do_xv_plot();
void xv_limit_calc(), do_window_care();
/*-----*/
void
scatter_plot(x, vx, xv_limits, n, plot_window, plot_picture, plot_label,
             time_test)
    register int n;
    register double *x, *vx;
    register max_min *xv_limits;
    WindowPtr plot_window;
    PicHandle *plot_picture;
    char *plot_label;
    Boolean time_test;
{
    register int i;
    Rect plot_rect;
    double deltax, deltay;
    char *e_string = NewPtr(256);
    SelectWindow(plot_window);
    ShowWindow(plot_window);
    SetPort(plot_window);
    ClipRect(&clipper);
    EraseRect(&clipper);
    plot_rect.top = 11;
    plot_rect.bottom = 261;
    plot_rect.left = 31;
    plot_rect.right = 481;
    limit_calc(x, vx, xv_limits, n);
    deltax = xv_limits[0].fmax - xv_limits[0].fmin;
    deltay = xv_limits[1].fmax - xv_limits[1].fmin;
    *plot_picture = OpenPicture(&big_rect);
    ShowPen();
    FrameRect(&plot_rect);
    for(i = 0; i < n; i++)
    {
        make_point((x[i] - xv_limits[0].fmin), (vx[i] - xv_limits[1].fmin),
                  deltax, deltay);
    }
    MoveTo(180, 295);
    DrawString(plot_label);
    if(time_test)
    {
        sprintf(e_string, "time = %.2lf", elapsed_time);
        CtoPstr(e_string);
        MoveTo(350, 276);
        DrawString(e_string);
    }
    ClosePicture();
    DisposPtr(e_string);
    SetWRefCon(plot_window, (long)*plot_picture);
}

```

```

do_window_care();
DrawControls(plot_window);
DrawGrowlcon(plot_window);
}
/*-----*/
void
do_xv_plot(x, vx, xv_limits, n, plot_window, plot_picture, plot_label,
           time_test)
{
    register int n;
    register double *x, *vx;
    register max_min *xv_limits;
    WindowPtr plot_window;
    PicHandle *plot_picture;
    char *plot_label;
    Boolean time_test;
    register int i;
    Rect plot_rect;
    double deltax, deltay;
    double dxdt = dx / dt;
    char *e_string = NewPtr(256);
    SelectWindow(plot_window);
    ShowWindow(plot_window);
    SetPort(plot_window);
    ClipRect(&clipper);
    EraseRect(&clipper);
    plot_rect.top = 11;
    plot_rect.bottom = 261;
    plot_rect.left = 31;
    plot_rect.right = 481;
    xv_limit_calc(x, vx, xv_limits, n);
    deltax = xv_limits[0].fmax - xv_limits[0].fmin;
    deltay = xv_limits[1].fmax - xv_limits[1].fmin;
    *plot_picture = OpenPicture(&big_rect);
    ShowPen();
    FrameRect(&plot_rect);
    ForeColor(redColor);
    for(i = 0; i < n; i++)
    {
        if(i > 127)
        {
            ForeColor(blueColor);
        }
        make_point((x[i] - .5 * vx[i] * dx - xv_limits[0].fmin),
                  (vx[i] * dxdt - xv_limits[1].fmin),
                  deltax, deltay);
    }
    ForeColor(blackColor);
    sprintf(e_string, "y min = %.2le", xv_limits[1].fmin);
    CtoPstr(e_string);
    MoveTo(38, 276);
    DrawString(e_string);
    sprintf(e_string, "y max = %.2le", xv_limits[1].fmax);
    CtoPstr(e_string);
    MoveTo(200, 276);
    DrawString(e_string);
    MoveTo(180, 295);
    DrawString(plot_label);
    if(time_test)

```

```

    {
        sprintf(e_string, "time = %.2lf", elapsed_time);
        CtoPtr(e_string);
        MoveTo(350, 276);
        DrawString(e_string);
    }
    ClosePicture();
    DisposPtr(e_string);
    SetWRefCon(plot_window, (long)*plot_picture);
    do_window_care();
    DrawControls(plot_window);
    DrawGrowlcon(plot_window);
}
/*-----*/
void
draw_plot(plot_window, plot_picture, plot_label)
    register WindowPtr plot_window;
    PicHandle plot_picture;
    char *plot_label;
{
    /*SetPort(plot_window);
    ClipRect(&clipper);
    EraseRect(&clipper);*/
    SetWRefCon(plot_window, (long)plot_picture);
    SetWTitle(plot_window, plot_label);
}
/*-----*/
void
limit_calc(f1, f2, f_limits, n)
    register double *f1, *f2;
    register max_min *f_limits;
{
    register int i;
    double f1l = 1.e10;
    double f1u = -1.e10;
    double f2l = 1.e10;
    double f2u = -1.e10;
    for (i = 0; i < n; i++)
    {
        if(f1[i] < f1l)
        {
            f1l = f1[i];
        }
        if(f1[i] > f1u)
        {
            f1u = f1[i];
        }
        if(f2[i] < f2l)
        {
            f2l = f2[i];
        }
        if(f2[i] > f2u)
        {
            f2u = f2[i];
        }
    }
    f_limits[0].fmax = f1u;
    f_limits[0].fmin = f1l;
    f_limits[1].fmax = f2u;
    f_limits[1].fmin = f2l;
}
/*-----*/
void
xv_limit_calc(f1, f2, f_limits, n)

```

```

register double *f1, *f2;
register max_min *f_limits;
{
    register int i;
    double f1l = 1.e10;
    double f1u = -1.e10;
    double f2l = 1.e10;
    double f2u = -1.e10;
    double g1, g2;
    double dxdt = dx / dt;
    for ( i = 0; i < n; i++)
    {
        g1 = f1[i] - .5 * f2[i] * dx;
        g2 = f2[i] * dxdt;
        if(g1 < f1l)
        {
            f1l = g1;
        }
        if(g1 > f1u)
        {
            f1u = g1;
        }
        if(g2 < f2l)
        {
            f2l = g2;
        }
        if(g2 > f2u)
        {
            f2u = g2;
        }
    }
    f_limits[0].fmax = f1u;
    f_limits[0].fmin = f1l;
    f_limits[1].fmax = f2u;
    f_limits[1].fmin = f2l;
}
/*-----*/
void
trace_plot(x, y, xy_limits, n, plot_window, plot_picture, plot_label,
           time_test)
{
    register double *x, *y;
    register max_min *xy_limits;
    char *plot_label;
    register int n;
    WindowPtr plot_window;
    PicHandle *plot_picture;
    Boolean time_test;
    register int i;
    Rect plot_rect;
    double deltax, deltay;
    char *e_string = NewPtr(256);
    SelectWindow(plot_window);
    ShowWindow(plot_window);
    SetPort(plot_window);
    ClipRect(&clipper);
    EraseRect(&clipper);
    plot_rect.top = 11;
    plot_rect.bottom = 261;
    plot_rect.left = 31;
    plot_rect.right = 481;
    deltax = xy_limits[0].fmax - xy_limits[0].fmin;
    deltay = xy_limits[1].fmax - xy_limits[1].fmin;

```

```

*plot_picture = OpenPicture(&big_rect);
ShowPen();
FrameRect(&plot_rect);
move_it(x[0] - xy_limits[0].fmin,
        y[0] - xy_limits[1].fmin, deltax, deltax);
for(i = 0; i < n; i++)
{
    line_it(x[i] - xy_limits[0].fmin,
            y[i] - xy_limits[1].fmin, deltax, deltax);
}
/*move_it(x[6] - xy_limits[0].fmin, 0., deltax, deltax);
line_it(x[6] - xy_limits[0].fmin, deltax, deltax, deltax);*/
sprintf(e_string, "y min = %.2le", xy_limits[1].fmin);
CtoPstr(e_string);
MoveTo(38, 276);
DrawString(e_string);
sprintf(e_string, "y max = %.2le", xy_limits[1].fmax);
CtoPstr(e_string);
MoveTo(200, 276);
DrawString(e_string);
MoveTo(180, 295);
DrawString(plot_label);
if(time_test)
{
    sprintf(e_string, "time = %.2lf", elapsed_time);
    CtoPstr(e_string);
    MoveTo(350, 276);
    DrawString(e_string);
}
ClosePicture();
DisposPtr(e_string);
SetWRefCon(plot_window, (long)*plot_picture);
do_window_care();
DrawControls(plot_window);
DrawGrowIcon(plot_window);
}
/*-----*/
void
move_it(x, y, dx, dy)
double x, y, dx, dy;
{
    int h, v;
    map_point(x, y, dx, dy, &h, &v);
    MoveTo(h, v);
}
/*-----*/
void
line_it(x, y, dx, dy)
double x, y, dx, dy;
{
    int h, v;
    map_point(x, y, dx, dy, &h, &v);
    LineTo(h, v);
}
/*-----*/
void
map_point(x, y, deltax, deltay, h, v)
double x, y;
double deltax, deltay;
register int *h, *v;

```

```

(    double tst;
    if(deltax > 1.e-45)
    {    x /= deltax;
        *h = ( x * 450);
    }
    else
    {    *h = 225;
    }
    *h += 31;
    if(deltay > 1.e-45)
    {    y /= deltax;
        *v = 250 - (y * 250);
    }
    else
    {    *v = 125;
    }
    *v += 11;
)
/*-----*/

void
make_point(x, y, deltax, deltax)
double x, y;
double deltax, deltax;
(    Rect pt_rect;
    int h, v;
    register int r = 1;
    map_point(x, y, deltax, deltax, &h, &v);
    pt_rect.top = v - r;
    pt_rect.left = h - r;
    pt_rect.bottom = v + r;
    pt_rect.right = h + r;
    PaintOval(&pt_rect);
)
/*-----*/

void
bin_fv(v, f_dist, vcenter, dv, n, nbins)
double vcenter, dv;
register double *f_dist, *v;
register int n, nbins;
(    register int i, j;
    double space = 2. * dv / nbins;
    double compare, com1;
    com1 = vcenter - space * (.5 * nbins - 1.);
    for(j = 0; j < nbins; j++)
    {    f_dist[j] = 0.;
    }
    for ( i = 0; i < n; i++)
    {    compare = com1;
        for(j = 0; j < nbins; j++)
        {    if(v[i] <= compare)
            {    f_dist[j] += 1.;
                break;
            }
            compare += space;
        }
    }
)
)

```


V. response.c

This module includes the routines that take care of all interactive mode events. In this mode menu selections (e.g. examining plot files) are executed and graphics viewing windows are moved and selected. The code for making a window "transparent" is found in these routines.

```

#include "plasma.h"
#define ENOUGH 3
#define NOT_ENOUGH 2

int DESKOPEN = 0;

extern WindowPtr draw_window[4];
extern PicHandle draw_picture[4];
extern FILE *output_file;

extern int my_ref;
extern SFRReply reply;
extern Rect drag_rect, grow_bounds, big_rect;
extern Boolean RETURN;
extern void do_new(), do_save_as(), do_open(), do_save();
extern void do_read(), handle_error(), switch_control();
extern do_print();

void do_controls(), do_menu(), do_about_menu(), do_files();
void do_mouse_down(), do_update(), do_examine(), move_bars();
void grow_window(), inval_grow(), do_activate(), show_scrolls();
void hide_scrolls(), do_window_care(), erase_grow(), valid_grow();
void toggle_see(), combine_pictures(), valid_scrolls(), grow_update();
void valid_rect(), define_grow(), do_see_update(), under_update();
Boolean MORE_PLOTS, INI, INF, see_through(), GROW, DRAG;
Boolean UNDER_ONLY;
PicHandle combo_pic;
Rect save_rect, overlap_rect, save_grow, save_v, save_h;
WindowPtr top_window, under_window;
RgnHandle draw_region;
Rect empty_rect = {0, 0, 0, 0};
RgnHandle empty_rgn;
/*-----*/
void
do_controls()
{
}
/*-----*/
void
do_menu(command)
long command;
{
    int menu_id = HiWord(command);
    int item = LoWord(command);
    char item_name[32];
    register int da_number;
    switch(menu_id)
    {
        case APPLE_MENU:
            if(item == ABOUT_ITEM)
            {
                do_about_menu();
            }
            else
            {
                GetItem(GetMHandle(menu_id), item, item_name);
                OpenDeskAcc(item_name);
                da_number = ((WindowPeek)FrontWindow())->windowKind;
                if(da_number < 0 && ++DESKOPEN == 1)
            }
        }
    }
}

```

```

        {      EnableItem(GetMHandle(EDIT_MENU), 0);
        }
    }
    break;
case FILE_MENU:
    do_files(item);
    break;
case EDIT_MENU:
    SystemEdit(item - 1);
    break;
case PLASMA_MENU:
    switch(item)
    {
        case INITIALIZE:
            break;
        case EXAMINE:
            MORE_PLOTS = TRUE;
            do_examine();
        case INTERRUPT:
            break;
        case TRANSPARENT:
            toggle_see();
            break;
        case RESTART:
            RETURN = TRUE;
            break;
    }
    }
    HilitMenu(0);
}
/*-----*/
void
do_about_menu()
{
}
/*-----*/
void
do_files(item)
int item;
{
    register int da_number = ((WindowPeek)FrontWindow())->windowKind;
    switch(item)
    {
        case PRINT:
            if(da_number >= 0)
            {
                do_print();
                if(PrintErr != 0)
                {
                    handle_error(PrintErr);
                }
            }
            break;
        case NEW:
            if(da_number >= 0)
            {
                do_new();
            }
            break;
        case OPEN:
            if(da_number >= 0)
            {
                do_open(draw_window[3], &draw_picture[3]);
            }
        }
    }
}

```

```

        )
        break;
    case CLOSE:
        if(da_number < 0)
        {
            CloseDeskAcc(da_number);
            if(--DESKOPEN <= 0)
            {
                DisableItem(GetMHandle(EDIT_MENU), 0);
            }
        }
        break;
    case SAVE:
    case SAVEAS:
        if(da_number >= 0)
        {
            do_save_as();
        }
        break;
    case QUIT:
        finish();
    default:
        break;
    }
}

/*-----*/
void
do_mouse_down(event)
    register EventRecord *event;
{
    register GrafPtr save_graf;
    Point b;
    WindowPtr mouse_window;
    register int da_number;
    int place_type = FindWindow(event->where, &mouse_window);
    register long getw = GetWRefCon(mouse_window);
    Boolean test;
    if(empty_rgn)
    {
        empty_rgn = NewRgn();
        RectRgn(empty_rgn, &empty_rect);
    }
    switch(place_type)
    {
        case inDesk:
            break;
        case inMenuBar:
            do_menu(MenuSelect(event->where));
            break;
        case inSysWindow:
            SystemClick(event, mouse_window);
            SystemTask();
            da_number = ((WindowPeek)FrontWindow())->windowKind;
            if(da_number >= 0 && --DESKOPEN <= 0)
            {
                DisableItem(GetMHandle(EDIT_MENU), 0);
            }
            break;
        case inContent:
            if( mouse_window != FrontWindow())
                SelectWindow(mouse_window);
            else
                do_controls(mouse_window, event->where);
    }
}

```

```

        break;
    case inDrag:
        if((test = (getw < 0 && mouse_window == FrontWindow())))
        {
            INI = see_through();
            save_rect = overlap_rect;
            DRAG = TRUE;
            if( INI )
            {
                InvalRect(&save_rect);
                valid_scrolls(mouse_window);
                valid_grow(mouse_window);
            }
        }
        DragWindow(mouse_window, event->where, &drag_rect);
        if(test)
        {
            INF = see_through();
            if((!INI) && INF && EmptyRgn(
                ((WindowPeek)mouse_window)->updateRgn))
            {
                InvalRect(&overlap_rect);
                UNDER_ONLY = TRUE;
            }
            if((!INF) && (!INI) && EmptyRgn(
                ((WindowPeek)mouse_window)->updateRgn))
            {
                DRAG = FALSE;
            }
        }
        break;
    case inGrow:
        if(test = (getw < 0 && mouse_window == FrontWindow()))
        {
            INI = see_through();
            GROW = TRUE;
            save_rect = overlap_rect;
        }
        grow_window(mouse_window, event->where);
        if(EmptyRgn(((WindowPeek)mouse_window)->updateRgn))
        {
            GROW = FALSE;
            break;
        }
        INF = see_through();
        break;
    case inGoAway:
        if (TrackGoAway(mouse_window, event->where))
        {
            HideWindow(mouse_window);
        }
        break;
    default:
        break;
}

/*-----*/
void
grow_update(window)
register WindowPtr window;
{
    GrafPtr save_graf;
    GetPort(&save_graf);
    SetPort(window);
    if(!draw_region)

```

```

    {
        draw_region = NewRgn();
    }
    RectRgn(draw_region, &overlap_rect);
    SectRgn(draw_region, ((WindowPeek)window)->updateRgn, draw_region);
    SetPort(save_graf);
}
/*-----*/
void
do_update(event)
{
    register EventRecord *event;
    GrafPtr save_graf;
    register WindowPtr update_window = (WindowPtr)event->message;
    register long getw = GetWRefCon(update_window);
    GetPort(&save_graf);
    SetPort(update_window);
    BeginUpdate(update_window);
    EraseRect(&update_window->portRect);
    DrawPicture((PicHandle)my_abs(getw), &big_rect);
    UpdtControl(update_window, update_window->visRgn);
    DrawGrowIcon(update_window);
    EndUpdate(update_window);
    SetPort(save_graf);
}
/*-----*/
void
do_see_update(event)
{
    register EventRecord *event;
    GrafPtr save_graf;
    register WindowPtr update_window = (WindowPtr)event->message;
    register long getw = GetWRefCon(update_window);
    Rect comp_rect;
    GetPort(&save_graf);
    SetPort(update_window);
    BeginUpdate(update_window);
    EraseRect(&update_window->portRect);
    DrawPicture((PicHandle)my_abs(getw), &big_rect);
    UpdtControl(update_window, update_window->visRgn);
    DrawGrowIcon(update_window);
    EndUpdate(update_window);
    SetPort(save_graf);
    if((INF = see_through()))
    {
        InvalRect(&overlap_rect);
        if(GROW)
        {
            ValidRect(&save_rect);
            /*SectRect(&save_grow, &overlap_rect, &save_grow);*/
            InvalRect(&save_grow);
            SectRect(&save_h, &overlap_rect, &save_h);
            InvalRect(&save_h);
            SectRect(&save_v, &overlap_rect, &save_v);
            InvalRect(&save_v);
        }
        valid_scroll(update_window);
        valid_grow(update_window);
        UNDER_ONLY = TRUE;
        if(EmptyRgn(((WindowPeek)update_window)->updateRgn))

```

```

        ( UNDER_ONLY = GROW = FALSE;
        )
    )
}
/*-----*/
void
under_update(event)
register EventRecord *event;
( Point origin;
  GrafPtr save_graf;
  register WindowPtr update_window = (WindowPtr)event->message;
  GetPort(&save_graf);
  BeginUpdate(update_window);
  /*EraseRect(&save_grow);*/
  SetPort(update_window);
  origin.h = origin.v = 0;
  LocalToGlobal(&origin);
  SetPort(top_window);
  GlobalToLocal(&origin);
  SetOrigin(-origin.h, -origin.v);
  DrawPicture((PicHandle)my_abs(GetWRefCon(update_window))), &big_rect);
  SetOrigin(0, 0);
  DrawPicture((PicHandle)my_abs(GetWRefCon(update_window))), &big_rect);
  EndUpdate(update_window);
  SetPort(save_graf);
  UNDER_ONLY = DRAG = GROW = FALSE;
)
/*-----*/

void
do_activate(event)
register EventRecord *event;
( register WindowPtr event_window = (WindowPtr)event->message;
  GrafPtr save_graf;
  if(activeFlag & event->modifiers)
  ( SetPort(event_window);
    if(GetWRefCon(event_window) < 0)
    ( INF = see_through();
      InvalRect(&overlap_rect);
      valid_scrolls(event_window);
    )
    show_scrolls(event_window);
  )
  else
  ( if(GetWRefCon(event_window) < 0)
    ( GetPort(&save_graf);
      SetPort(event_window);
      InvalRect(&overlap_rect);
      valid_scrolls(event_window);
      SetPort(save_graf);
    )
    hide_scrolls(event_window);
  )
  DrawGrowIcon(event_window);
  /*valid_grow(event_window);*/
)

```

```

/*-----*/
void
valid_rect(window, bad_rect)
register WindowPtr window;
register Rect *bad_rect;
{
    register ControlHandle control = ((WindowPeek)window)->controlList;
    register long label;
    Rect val_rect;
    while(control)
    {
        label = GetCRefCon(control);
        if(label == (long)V_SCROLL || label == (long)H_SCROLL)
        {
            SectRect(&(*control)->controlRect, bad_rect, &val_rect);
            ValidRect(&val_rect);
        }
        control = (*control)->nextControl;
    }
    /*define_grow(window, &val_rect);
    SectRect(&val_rect, bad_rect, &val_rect);
    ValidRect(&val_rect);*/
}
/*-----*/

void
valid_scrolls(window)
register WindowPtr window;
{
    register ControlHandle control = ((WindowPeek)window)->controlList;
    register long label;
    while(control)
    {
        label = GetCRefCon(control);
        if(label == (long)V_SCROLL || label == (long)H_SCROLL)
        {
            ValidRect(&(*control)->controlRect);
        }
        control = (*control)->nextControl;
    }
}
/*-----*/

void
show_scrolls(window)
register WindowPtr window;
{
    register ControlHandle control = ((WindowPeek)window)->controlList;
    register long label;
    while(control)
    {
        label = GetCRefCon(control);
        if(label == (long)V_SCROLL || label == (long)H_SCROLL)
        {
            HiliteControl(control, 0);
            ValidRect(&(*control)->controlRect);
        }
        control = (*control)->nextControl;
    }
}
/*-----*/

void
hide_scrolls(window)
register WindowPtr window;
{
    register ControlHandle control = ((WindowPeek)window)->controlList;
    register long label;

```



```

while(control)
{
    label = GetCRefCon(control);
    if(label == (long)V_SCROLL || label == (long)H_SCROLL)
    {
        HiliteControl(control, 255);
        ValidRect(&(*control)->controlRect);
    }
    control = (*control)->nextControl;
}
)

/*-----*/
void
moveBars(window)
register WindowPtr window;
{
    register WindowPeek peek = (WindowPeek)window;

    register ControlHandle control = peek->controlList;
    int new_top = window->portRect.top;
    int new_left = window->portRect.left;
    int new_bottom = window->portRect.bottom;
    int new_right = window->portRect.right;
    register long label;

    while (control)
    {
        label = GetCRefCon(control);
        if (label == (long)V_SCROLL)
        {
            HideControl(control);
            save_v = (*control)->controlRect;
            MoveControl(control, new_right - BAR_WIDTH, new_top - 1);
            SizeControl(control, 16, new_bottom - new_top - 13);
            ShowControl(control);
            ValidRect(&(*control)->controlRect);
        }
        else if (label == (long)H_SCROLL)
        {
            HideControl(control);
            save_h = (*control)->controlRect;
            MoveControl(control, new_left - 1, new_bottom - BAR_WIDTH);
            SizeControl(control, new_right - new_left - 13, 16);
            ShowControl(control);
            ValidRect(&(*control)->controlRect);
        }
        control = (*control)->nextControl;
    }
}

/*-----*/
void
growWindow(window, mouse_point)
register WindowPtr window;
Point mouse_point;
{
    long new_bounds;
    new_bounds = GrowWindow(window, mouse_point, &grow_bounds);
    if(new_bounds != 0)
    {
        invalGrow(window);
        /*EraseRect(&save_grow);*/
        SizeWindow(window, LoWord(new_bounds), HiWord(new_bounds), TRUE);
    }
}

```

```

        move_bars(window);
        DrawGrowlcon(window);
        valid_grow(window);
    )
}
/*-----*/
void
define_grow(window, grow_rect)
WindowPtr window;
register Rect *grow_rect;
(
    Rect port_rect;
    port_rect = window->portRect;
    SetRect(grow_rect, port_rect.right - 16, port_rect.bottom - 16,
            port_rect.right, port_rect.bottom);
)
/*-----*/
void
inval_grow(window)
WindowPtr window;
(
    Rect temp_rect;
    define_grow(window, &temp_rect);
    InvalRect(&temp_rect);
    save_grow = temp_rect;
)
/*-----*/
void
valid_grow(window)
WindowPtr window;
(
    Rect temp_rect;
    define_grow(window, &temp_rect);
    ValidRect(&temp_rect);
)
/*-----*/
void
do_examine()
(
    int response;
    static int window_number = 0;
    int selection;
    register WindowPtr the_window;
    if(my_ref == -999)
    (
        selection = window_number % 4;
        do_open(draw_window[selection], &draw_picture[selection]);
        /*SelectWindow(draw_window[selection]);*/
        window_number++;
    )
    while(MORE_PLOTS)
    (
        the_window = FrontWindow();
        hide_scrolls(the_window);
        erase_grow(the_window);
        response = Alert(257, 0L);
        show_scrolls(the_window);
        DrawGrowlcon(the_window);
        switch(response)
        (
            case ENOUGH:
                MORE_PLOTS = FALSE;
                selection = window_number % 4;

```

```

do_open(draw_window[selection], &draw_picture[selection] );
/*SelectWindow(draw_window[selection]);*/
do_window_care();
window_number++;
break;
case NOT_ENOUGH:
    selection = window_number % 4;
    /*SelectWindow(draw_window[selection]);*/
    do_read(draw_window[selection],
            &draw_picture[selection], reply, my_ref);
    do_window_care();
    window_number++;
    break;
    )
}
}
/*-----*/
void
do_window_care()
(
    Boolean activate = TRUE;
    EventRecord this_event;
    while(activate)
    (
        EventAvail(everyEvent, &this_event);
        if(this_event.what == activateEvt)
        (
            GetNextEvent(everyEvent, &this_event);
            do_activate(&this_event);
        )
        else
        (
            activate = FALSE;
        )
    )
)

void
erase_grow(window)
register WindowPtr window;
(
    Rect temp_rect, port_rect;
    port_rect = window->portRect;
    SetRect(&temp_rect, port_rect.right - 14, port_rect.bottom - 14,
            port_rect.right, port_rect.bottom);
    EraseRect(&temp_rect);
)
/*-----*/
void
toggle_see()
(
    top_window = FrontWindow();
    SetWRefCon(top_window, -GetWRefCon(top_window));
    if((UNI = INF = see_through()))
    (
        InvalRect(&overlap_rect);
        valid_scrolls(top_window);
        valid_grow(top_window);
        if(GetWRefCon(top_window) < 0)
        (
            UNDER_ONLY = TRUE;
        )
    )
)

```

```

/*-----*/
Boolean
see_through()
(
    GrafPtr save_graf;
    Boolean IN = TRUE;
    top_window = FrontWindow();
    under_window = (WindowPtrX(((WindowPeek)top_window)->nextWindow);
    GetPort(&save_graf);
    (
        SetPort(under_window);
        LocalToGlobal((Point *)&under_window->portRect);
        LocalToGlobal((Point *)&((under_window->portRect).bottom));
        SetPort(top_window);
        LocalToGlobal((Point *)&top_window->portRect);
        LocalToGlobal((Point *)&((top_window->portRect).bottom));
        SectRect(&(top_window->portRect), &(under_window->portRect),
            &overlap_rect);
        if(((long *)&overlap_rect)[0] == 0L && ((long *)&overlap_rect)[1] == 0L)
        {
            IN = FALSE;
        }
        GlobalToLocal((Point *)&top_window->portRect);
        GlobalToLocal((Point *)&((top_window->portRect).bottom));
        GlobalToLocal((Point *)&overlap_rect);
        GlobalToLocal((Point *)&overlap_rect.bottom);
        SetPort(under_window);
        GlobalToLocal((Point *)&under_window->portRect);
        GlobalToLocal((Point *)&((under_window->portRect).bottom));
        SetPort(save_graf);
    )
    return( IN );
)

```

VI. advance.c

This module contains the heart of the simulation program. The routines that carry out the "leap frog" advancement of the simulation particles in time and the routines that calculate and advance the field quantities are in this module.

```

#include "plasma.h"

extern double *x, *vx, *vy;
extern double aei, epsi, rho0, rho[NG1M], phi[NG1M], e[NG1M], eo, wo;
extern int it, ith1, irho, irhos, iphi, ie, ixvx, ivxvy, ifvx;
extern double elapsed_time;
extern double scratch[NG1M], ffit[NGTWO];
extern int ng, iw;
extern double L, dx, dt;
extern double ese[NTH1], nms[NSPM];
extern double *esem[MMAX];
extern int mplot[MMAX];
extern double x_points[NGMAX];
extern WindowPtr draw_window[4];
extern PicHandle draw_picture[4];
extern      int rho_file, rhos_file, e_file, phi_file, vxvy_file, vxv_file,
              fv_file;
void get_fields(), transform_rho(), rho_to_phi(), save_modes(), do_inverse();
void electric_field(), reset_rho(), accelerate(), nearest_point();
void p_conserve(), e_conserve(), advance();

extern void field_init(), sincos2(), inverse_sincos2(), save_graph();
extern void trace_plot(), show();

extern FILE *output_file;
/*-----*/
void
get_fields(ith)
{
    register int ith;
    static int ng2 = 0;
    static double sm[NG2M], ksqi[NG2M];
    static max_min e_limits[2];
    static max_min rho_limits[2];
    static max_min phi_limits[2];
    double Li;
    register int i;
    if(ng2)
    {
        ng2 = ng/2;
        field_init(sm, ksqi, ng2);
    }
    rho[0] += rho[ng];
    rho[ng] = rho[0];
    if(irho != 0 && (ith % irho == 0))
    {
        save_graph(rho_file, x_points, rho, rho_limits, ng, draw_window[0],
                  &draw_picture[0], "pCharge Density", TRUE, trace_plot);
    }
    transform_rho();
    rho_to_phi(ith, ng2, ksqi, sm);
    save_modes(ith);
    do_inverse();
    if(irhos != 0 && (ith % irhos == 0))
    {
        save_graph(rhos_file, x_points, rho, rho_limits, ng, draw_window[0],
                  &draw_picture[0], "pSmoothed Charge Density", TRUE, trace_plot);
    }
    if(iph1 != 0 && (ith % iphi == 0))
    {
        save_graph(phi_file, x_points, phi, phi_limits, ng, draw_window[1],

```

```

                                &draw_picture[1], "\pPotential Phi", TRUE, trace_plot);
    }
    electric_field();
    if(!ie || (ith % ie == 0))
    {
        save_graph(e_file, x_points, e, e_limits, ng, draw_window[2],
                    &draw_picture[2], "\pElectric Field", TRUE, trace_plot);
    }
    reset_rho();
    fprintf(output_file, "\nreset_rho");
    show(x_points, rho, ng + 1);
    ael = 1.;
}
/*-----*/
void
transform_rho()
{
    register int i;
    double hdx;
    register double *rhok = rho;
    hdx = .5 * dx;
    for( i = 0; i < ng; i++)
    {
        rho[i] *= hdx;
        scratch[i] = 0.;
    }
    sincos2(rho, scratch, fft1, rhok, scratch, ng);
    rhok[0] = 0.;
}
/*-----*/
void
rho_to_phi(ith, ng2, ksqi, sm)
{
    int ith, ng2;
    register double *ksqi;
    double *sm;
    double eses = 0.;
    register int i, index;
    register double *phik = phi;
    register double *rhok = rho;
    phik[0] = 0.;
    index = ng - 1;
    for( i = 1; i < ng2; i++, index--)
    {
        phik[i] = ksqi[i - 1] * rhok[i];
        phik[index] = ksqi[i - 1] * rhok[index];
        eses += rhok[i] * phik[i] + rhok[index] * phik[index];
        rhok[i] *= sm[i - 1];
        rhok[index] *= sm[i - 1];
    }
    phik[ng2] = ksqi[ng2 - 1] * rhok[ng2];
    eses[ith] = (2. * eses + rhok[ng2] * phik[ng2]) / (2. * L);
    rhok[ng2] *= sm[ng2 - 1];
}
/*-----*/
void
save_modes(ith)
{
    register int ith;
    register int i = 0;
    register int index = ng;
    double temp;

```

```

    register double *phik = phi;
    register double *rhok = rho;
    while((mplot[i] != 0) && (i <= MMAX))
    {
        temp = (rhok[i] * phik[i] + rhok[index] * phik[index]) / L;
        if(i == index)
        {
            temp *= .25;
        }
        esem[i+1][ith] = temp;
        index--;
    }
}
/*-----*/

void
do_inverse()
{
    register double *phik = phi;
    register double *rhok = rho;
    double Li = 1. / L;
    register int i;
    for( i = 0; i < ng; i++)
    {
        rhok[i] *= Li;
        phik[i] *= Li;
    }
    inverse_sincos2(phik, rhok, fftl, phi, rho, ng);
    phi[ng] = phi[0];
    rho[ng] = rho[0];
/*    fprintf(output_file, "\ndo_inverse");
    show(x_points, rho, ng + 1);*/
}
/*-----*/

void
electric_field()
{
    double e0t, hdx1, dxi;
    register int i;
    e0t = e0 * cos(w0 * elapsed_time);
    switch(iw)
    {
        case ZERO_ORDER:
        case MOMENTUM:
            hdx1 = .5 / dx;
            for (i = 1; i < ng; i++)
            {
                e[i] = (phi[i-1] - phi[i + 1]) * hdx1 + e0t;
            }
            e[0] = (phi[ng - 1] - phi[1]) * hdx1 + e0t;
            e[ng] = e[0];
            break;
        case ENERGY:
            dxi = 1. / dx;
            for (i = 0; i < ng; i++)
            {
                e[i] = (phi[i] - phi[i + 1]) * dxi + e0t;
            }
            e[ng] = e[0];
            break;
    }
}
/*-----*/

void
reset_rho()

```



```

(    register int i;
    for ( i = 0; i < ng; i++)
    {    rho[i] = rho0;
    }
    rho[ng] = 0.;
)
/*-----*/

void
accelerate(il, iu, q, m, t, p, ke)
double q, m, t;
register double *p, *ke;
register int il, iu;
(    register double *a = e;
    double dxdt = dx / dt;
    double ae = (q / m) * (dt / dxdt);
    double temp;
    register int i;
    if(NEED_ROTATION)
    {    ae *= .5;
    }
    if(ae != ael)
    {    temp = ae / ael;
        for( i = 0; i <= ng; i++)
        {    a[i] *= temp;
        }
        ael = ae;
    }
)
switch(iw)
(    case ZERO_ORDER:
        nearest_point(il, iu, dxdt, m, ke, p);
        break;
    case MOMENTUM:
        p_conserve(il, iu, dxdt, m, t, ke, p);
        break;
    case ENERGY:
        e_conserve(il, iu, dxdt, m, ke, p);
        break;
)
/*-----*/

void
nearest_point(il, iu, dxdt, m, ke, p)
register int il, iu;
double dxdt, m;
register double *ke, *p;
(    double v1s = 0.;
    double v2s = 0.;
    register double *a = e;
    double v0, vn;
    int i, j;
    for( i = il; i < iu; i++)
    {    j = (x[i] + .5);
        v0 = vx[i];
        vn = v0 + a[j];
        v1s += vn;
        v2s += vn * v0;
    }
)

```

```

        vx[i] = vn;
    }
    *p += m * vls * dxdt;
    *ke += .5 * m * v2s * dxdt * dxdt;
}
/*-----*/
void
p_conserve(il, iu, dxdt, m, t, ke, p)
register int il, iu;
double dxdt, m, t;
register double *p, *ke;
{
    register int i, j;
    double v0, vn, vls, v2s, s, aa, vxx, vyy;
    register double *a = e;
    if(NEED_ROTATION)
    {
        s = 2. * t / (1. + t * t);
        v2s = 0.;
        for( i = il; i < iu; i++)
        {
            j = x[i];
            aa = a[j] + (x[i] - j) * (a[j + 1] - a[j]);
            vyy = vy[i];
            vxx = vx[i] - t * vyy + aa;
            vyy += s * vxx;
            vxx -= t * vyy;
            v2s += vxx * vxx + vyy * vyy;
            vx[i] = vxx + aa;
            vy[i] = vyy;
        }
        *ke += .5 * m * v2s * dxdt * dxdt;
    }
    else
    {
        vls = 0.;
        v2s = 0.;
        for( i = il; i < iu; i++)
        {
            j = x[i];
            v0 = vx[i];
            vn = v0 + a[j] + (x[i] - j) * (a[j + 1] - a[j]);
            vls += vn;
            v2s += v0 * vn;
            vx[i] = vn;
        }
        *p += m * vls * dxdt;
        *ke += .5 * m * v2s * dxdt * dxdt;
    }
}
/*-----*/
void
e_conserve(il, iu, dxdt, m, ke, p)
register int il, iu;
double dxdt, m;
register double *p, *ke;
{
    register int i, j;
    double v0, vn, vls, v2s;
    register double *a = e;
    vls = v2s = 0.;
    for( i = il; i < iu; i++)

```

```

    {
        j = x[i];
        v0 = vx[i];
        vn = v0 + a[j];
        v1s += vn;
        v2s += vn * v0;
        vx[i] = vn;
    }
    *p += m * v1s * dxdt;
    *ke += .5 * m * v2s * dxdt * dxdt;
}
/*-----*/

void
advance(il, iu, q)
register int il, iu;
double q;
{
    register int i, j;
    double qdx = q / dx;
    double xn = ng;
    double drho;
    switch(iw)
    {
        case ZERO_ORDER:
            for( i = il; i < iu; i++)
            {
                x[i] += vx[i];
                if(x[i] < 0.)
                {
                    x[i] += xn;
                }
                if(x[i] >= xn)
                {
                    x[i] -= xn;
                }
                j = x[i] + .5;
                rho[j] += qdx;
            }
            break;
        case MOMENTUM:
        case ENERGY:
            for( i = il; i < iu; i++)
            {
                x[i] += vx[i];
                if(x[i] < 0.)
                {
                    x[i] += xn;
                }
                if(x[i] >= xn)
                {
                    x[i] -= xn;
                }
                j = x[i];
                drho = qdx * (x[i] - j);
                rho[j] += qdx - drho;
                rho[j + 1] += drho;
            }
            break;
    }
}

```

VII. `fourier.c`

This module consists of the fast Fourier transform routines utilized in `advance.c`.

```

#include "plasma.h"
double sines[15], sinep[15];
void fill_sine_table(), fourl(), sincos2(), inverse_sincos2();
void
fill_sine_table()
{
    double t = atan(1.);
    register int i;
    sines[0] = 1.;
    for(i = 1; i < 15; i++)
    {
        sines[i] = sin(t);
        sinep[i] = -2. * pow(sines[i], 2.);
        t *= .5;
    }
}

void
fourl(data, nn, isign)
register double *data;
int nn, isign;
{
    register int i;
    register int j = 0;
    int n = 2 * nn;
    register int mmax, m, istep;
    int sincnt;
    double wtemp, wr, wpr, wpi, wi, tempr, tempi;
    for(i = 0; i < n; i += 2)
    {
        if(j > i)
        {
            tempr = data[j];
            tempi = data[j + 1];
            data[j] = data[i];
            data[j + 1] = data[i + 1];
            data[i] = tempr;
            data[i + 1] = tempi;
        }
        m = nn;
        while( (m >= 2) && ((j + 1) > m) )
        {
            j -= m;
            m /= 2;
        }
        j += m;
    }
    mmax = 2;
    wpi = 0.;
    sincnt = 0;
    while(mmax < n)
    {
        istep = 2 * mmax;
        wpr = sinep[sincnt];
        wr = 1.;
        wi = 0.;
        for(m = 0; m < mmax; m += 2)
        {
            for(i = m; i < n; i += istep)
            {
                j = i + mmax;
                tempr = wr * data[j] - wi * data[j + 1];
                tempi = wr * data[j + 1] + wi * data[j];
                data[j] = data[i] - tempr;
                data[j + 1] = data[i + 1] - tempi;
                data[i] += tempr;
            }
        }
    }
}

```

```

        data[i + 1] += temp1;
    }
    wtemp = wr;
    wr += wr * wpr - wi * wpi;
    wi += wi * wpr + wtemp * wpi;
}
mmax *= 2;
wpi = (isign > 0 ? sines[sincnt] : -sines[sincnt]);
sincnt++;
}
}

void
sincos2(data1, data2, fft, datak1, datak2, n)
double *data1, *data2;
register double *datak1, *datak2;
register double (*fft)(2);
register int n;
{
    double rp, ip, rm, im;
    register int j, index;
    for(j = 0; j < n; j++)
    {
        fft[j][0] = data1[j];
        fft[j][1] = data2[j];
    }
    four1(fft, n, 1);
    datak1[0] = 2. * fft[0][0];
    datak2[0] = 2. * fft[0][1];
    for(j = 1, index = n - 1; j < index; j++, index--)
    {
        rp = fft[j][0];
        ip = fft[j][1];
        rm = fft[index][0];
        im = fft[index][1];
        datak1[j] = (rp + rm);
        datak2[j] = (ip + im);
        datak1[index] = (ip - im);
        datak2[index] = (rm - rp);
    }
    datak1[j] = 2. * fft[j][0];
    datak2[j] = 2. * fft[j][1];
}

void
inverse_sincos2(datak1, datak2, fft, data1, data2, n)
double *data1, *data2;
register double *datak1, *datak2;
register double (*fft)(2);
register int n;
{
    double ca, sb, cb, sa;
    register int j, index;
    fft[0][0] = datak1[0];
    fft[0][1] = datak2[0];
    for(j = 1, index = n - 1; j < index; j++, index--)
    {
        ca = datak1[j];
        cb = datak2[j];
        sa = datak1[index];
        sb = datak2[index];
        fft[j][0] = ca - sb;
        fft[j][1] = cb + sa;
    }
}

```

```
fft[indexX0] = ca + sb;  
fft[indexX1] = cb - sa;  
}  
fft[jX0] = data1[j];  
fft[jX1] = data2[j];  
four1(fft, n, -1);  
for(j = 0; j < n; j++)  
(  
    data1[j] = fft[jX0];  
    data2[j] = fft[jX1];  
)  
)
```

VIII. plasma.h

This is the header file that contains definitions of various quantities used in the other modules. Many of these definitions are in the form of include statements for still other header files, mainly

Macintosh system and i/o definitions.


```

#include "math.h"
#include "stdio.h"
#include "unix.h"
#include "sane.h"
#include "QuickDraw.h"
#include "MemoryMgr.h"
#include "MacTypes.h"
#include "WindowMgr.h"
#include "ControlMgr.h"
#include "EventMgr.h"
#include "DeskMgr.h"
#include "MenuMgr.h"
#include "ToolboxUtil.h"
#include "DialogMgr.h"
#include "ResourceMgr.h"
#include "FontMgr.h"
#include "TextEdit.h"
#include "PrintMgr.h"
#include "FileMgr.h"
#include "StdFilePkg.h"

#define my_abs(x) ((x) > 0? (x) : -(x))
#define my_max(x,y) ((x) > (y)? (x) : (y))

#define NTH 500
#define MMAX 16
#define NSPM 3
#define NGMAX 256
#define NG1M (NGMAX + 1)
#define NG2M (NGMAX / 2)
#define NGTWO (2 * NGMAX)
#define NPAR 512
#define NTH1 (NTH + 1)
#define NTH2 (NTH + 2)
#define NSPM1 (NSPM + 1)

#define ZERO_ORDER 1
#define MOMENTUM 2
#define ENERGY 3

#define ORDERED (vt2 != 0.)
#define MAGNETIZED (wc != 0.)
#define ANOTHER (nlg != 0)
#define RANDOM (vt1 != 0.)
#define FIRST_GROUP (il == 0)
#define NEED_ROTATION (t != 0.)

#define V_SCROLL 256
#define H_SCROLL 257
#define BAR_WIDTH 15
#define GRAPH_WINDOW 256

#define APPLE_MENU 1
#define FILE_MENU 256
#define PLASMA_MENU 257
#define EDIT_MENU 258

```

```
*define ABOUT_ITEM 1

*define NEW 1
*define OPEN 2
*define CLOSE 3
*define SAVE 4
*define SAVEAS 5
*define REVERT 6
*define PRINT 8
*define QUIT 10

*define ENDALERT 256

*define INITIALIZE 1
*define EXAMINE 2
*define INTERRUPT 3
*define TRANSPARENT 4
*define RESTART 5

*define DRVR 0x44525652L /* The string "DRVR" as a long */
*define mk_long(x) (*((long *) & (x)))

typedef struct(double fmax,
               fmin;) max_min;
```

9.0 PROBLEMS ENCOUNTERED DURING REPORTING PERIOD, R.D. ESTES, PI

None

10.0 ACTIVITY PLANNED FOR NEXT REPORTING PERIOD, R.D. ESTES, PI

The work of this task is proceeding on schedule. In the coming period the software development will continue with the modification of the plasma simulation program to more closely approximate the physics of our hollow cathode plasmas. This means that collisional effects will have to be added and that the boundary conditions will have to be modified. Initial computer experiments will be made and their results compared with actual laboratory results. Parameters will be varied to determine which ones are most important. In addition to continuing the line of development already begun, we will be investigating the use of another computer code for applicability to our problems. MASK, a two-dimensional, fully electromagnetic, relativistic plasma simulation code developed at MIT and improved by Scientific Applications International Corp. (SAIC) is available for our use. The relativistic features are clearly not needed and would carry a substantial overhead, but it may be possible to modify the code easily to run in a nonrelativistic mode. A CONVEX computer is now available to us in the Radio and Geoastronomy Division of the Harvard-Smithsonian Center for Astrophysics. The vectorizing capabilities of this machine may make simulations with larger

numbers of particles practical.

Page 120